Réorganisation hiérarchique de visualisations dans OLAP

Sébastien Lafon*, Fatma Bouali**,*, Christiane Guinot***,*, Gilles Venturini*

* Université François-Rabelais de Tours, Laboratoire d'Informatique
64 avenue Jean Portalis, 37200 Tours, France,
venturini@univ-tours.fr
** Université de Lille2, IUT, Dpt STID
25-27 Rue du Maréchal Foch, 59100 Roubaix, France
fatma.bouali@univ-lille2.fr
*** CE.R.I.E.S.
20 Rue Victor Noir, 92521 Neuilly-sur-Seine, France
christiane.guinot@ceries-lab.com

Résumé. Dans cet article nous proposons un nouvel algorithme pour la réorganisation hiérarchique des cubes OLAP (On-Line Analytical Processing) ayant pour objectif d'améliorer leur visualisation. Cet algorithme se caractérise par le fait qu'il peut traiter des dimensions organisées hiérarchiquement et optimiser conjointement les dimensions du cube, contrairement aux autres approches. Il utilise un algorithme génétique qui réorganise des arbres n-aires quelconques. Il a été intégré dans une interface OLAP puis testé en comparaison avec d'autres approches de réorganisation, et fournit des résultats très positifs. A ce titre, nous avons également généralisé l'algorithme heuristique classique BEA ("bond energy algorithm") au cas de hiérarchies OLAP. Enfin, notre approche a été évaluée par des utilisateurs et les résultats soulignent l'intérêt de la réorganisation dans des exemples de tâches à résoudre pour OLAP.

1 Introduction

Dans beaucoup de visualisations, des signes ou éléments visuels doivent être placés dans un espace 1D, 2D ou 3D afin d'améliorer la compréhensibilité des informations visualisées. Si l'on s'intéresse aux réorganisations unidimensionnelles, des méthodes ont été développées pour réordonner par exemple l'ordre des axes dans les coordonnées parallèles (Ankerst et al., 1998) (Yang et al., 2003). Ce problème de réorganisation unidimensionnelle s'apparente souvent au problème du voyageur de commerce (Ankerst et al., 1998) (Climer et Zhang, 2006) : les éléments visuels e_i à ordonner représentent des villes et les similarités entre les e_i représentent les distances entre les villes. En ce qui concerne la réorganisation de matrices, de nombreux travaux ont eu lieu en visualisation (voir un survol dans (Fekete et Henry, 2009) (Liiv, 2010) (Garcia et al., 2010)), avec par exemple des approches interactives comme dans (Bertin, 1981). Cependant, il a été montré dans plusieurs études (voir survol dans (Garcia et al., 2010)) que les utilisateurs préférent des approches automatiques. Dans (McCormick et al., 1972) est présenté l'algorithme BEA (Bond Energy Algorithm) qui utilise une heuristique pour réorganiser

les lignes (respectivement les colonnes) d'une matrice. BEA choisit au hasard une ligne (respectivement une colonne), puis insère les autres lignes (respectivement colonnes) à gauche, à droite ou au milieu de celles déjà positionnées, selon l'emplacement qui fait augmenter le plus un critère d'efficacité ("measure of effectiveness"). Notons que dans la section 4, nous étendrons cet algorithme aux cas des dimensions hiérarchiques. D'autres méthodes peuvent être citées comme par exemple (Caraux, 1984) (Climer et Zhang, 2006). Dans cet article nous allons nous intéresser particulièrement à une dernière catégorie de problèmes dans le cas où les éléments à réorganiser sont structurés en une hiérarchie. Un problème classique de ce type correspond à la réorganisation des dendogrammes générés par des méthodes de classification ascendante hiérarchique (CAH), comme dans (Vandev et Tsvetanova, 1997), dans (Bar-joseph et al., 2001) avec un algorithme optimal de complexité temporelle en $O(n^4)$, ou encore dans (Morris et al., 2003) avec un algorithme utilisant le recuit simulé. Ces approches sont cependant limitées à des arbres binaires. Dans (Bar-joseph et al., 2003) est présenté une généralisation de (Bar-joseph et al., 2001) pouvant réorganiser les arbres n-aires. C'est, à notre connaissance, l'un des rares algorithmes de réorganisation visuelle hiérarchique pouvant traiter des arbres n-aires. Cet algorithme s'appuie sur une transformation de l'arbre n-aire en arbre binaire en ajoutant des noeuds fictifs. Sa complexité temporelle est élevée ($O(4^{k+o(k)}n^3)$ où k représente l'arité maximale de l'arbre) dans le cas général, car cet algorithme reste lié à une forme particulière d'arbres de classification dans lesquels k reste faible. Il traite l'optimisation d'un seul arbre à la fois et ne peut donc pas optimiser les dimensions de manière conjointe. Il n'y a donc pas à notre connaissance d'algorithmes de réorganisation visuelle pour les arbres n-aires au sens large et qui puissent également optimiser plusieurs dimensions de manière conjointe (optimisation globale de la structure visuelle).

Dans OLAP (Codd et al., 1993) (Chaudhuri et Dayal, 1997), l'utilisateur analyse visuellement et interactivement des données multidimensionnelles (voir survol des visualisations 2D dans (Cuzzocrea et Mansmann, 2009)), où les dimensions contiennent des membres, ces membres pouvant être organisés en niveaux hiérarchiques. A l'intersection de chaque dimension sont définies une ou plusieurs valeurs de mesures. OLAP met à la disposition de l'expert du domaine plusieurs opérateurs permettant d'interagir avec l'hypercube et d'explorer interactivement les données. Il existe notamment des opérateurs dont le but est de réorganiser l'ordre des membres sur une dimension (exemple de l'opérateur Switch). Les cubes OLAP sont donc particulièrement concernés par le problème de la réorganisation, car en réorganisant les dimensions on peut faire apparaître des régularités importantes pour l'expert du domaine (des groupes de données similaires, des données aberrantes, etc). Cependant, réorganiser un cube OLAP de manière "manuelle" représente une tâche lourde pour l'utilisateur. Comme nous allons le montrer dans cet article, des approches automatiques ont été étudiées pour réorganiser un cube OLAP, mais elles restent encore peu nombreuses et sont limitées principalement du fait qu'elles ne traitent pas les dimensions organisées hiérarchiquement. Pour cela nous devons étudier des algorithmes de réorganisation hiérarchique s'appliquant sur des arbres n-aires quelconques.

La suite de l'article est organisée de la manière suivante : dans la section 2, nous présentons un état de l'art du domaine de la réorganisation visuelle de cubes OLAP. Dans la section 3 nous présentons notre algorithme génétique (AG) et ses différents éléments. Dans la section 4 nous proposons des algorithmes d'optimisation niveau par niveau. Dans la section 5 nous détaillons les résultats expérimentaux obtenus lors d'une comparaison avec d'autres approches et lors

d'une évaluation utilisateur. Dans la section 6 nous présentons les conclusions et perspectives.

2 Réorganisation des visualisations dans OLAP

Avant de décrire les approches de réorganisation visuelle dans OLAP, notons que des approches de réorganisation de cubes OLAP ont été proposées dans le cadre de la compression de données comme par exemple (Barbará et Sullivan, 1997), mais que ces approches n'ont pas d'objectifs liés à la visualisation. En ce qui concerne les approches de réorganisation visuelle pour OLAP, nous pouvons citer (Choong et al., 2003) où les auteurs cherchent une représentation parfaite du cube sous la forme d'une matrice de Robinson (valeur de similarité décroissante à partir de la diagonale). Dans (Ben Messaoud et al., 2004), un nouvel opérateur OLAP est présenté, portant le nom d'OpAC (Opérateur d'Agrégation par Classification), et qui utilise la CAH pour construire une hiérarchie (sans tenir compte d'une éventuelle hiérarchie existante). Dans (Ben Messaoud et al., 2007), une méthode se basant sur l'Analyse des Correspondances Multiples est présentée pour améliorer la visualisation du cube. Cette méthode de réorganisation a pour avantage de réorganiser toutes les dimensions simultanément mais elle ne prend pas en compte la hiérarchie de celles-ci. Dans (Sureau et al., 2008), nous avons présenté un premier algorithme génétique et qui ne prenait pas en compte la hiérarchie des dimensions et ayant des temps d'exécution trop longs pour OLAP (15 à 40 minutes).

L'algorithme génétique que nous avons conçu afin de réorganiser un cube OLAP se distingue des autres approches de réorganisation de la manière suivante : il traite les dimensions organisées hiérarchiquement (contrairement aux approches précédentes pour OLAP), sans faire d'hypothèses sur la forme de l'arbre (contrairement à (Bar-joseph et al., 2003) ou aux approches ne traitant que les arbres binaires). Notre méthode concerne des cubes de données affichant jusqu'à trois dimensions et deux mesures, alors que les approches précédentes se limitaient à deux dimensions et une mesure. Contrairement à beaucoup d'algorithmes de réorganisation, notre approche peut réorganiser les trois dimensions simultanément, et non plus isolément, de manière à optimiser globalement l'aspect du cube et permettre l'utilisation de fonctions de coût non séparables par dimension. Notre approche prend en compte la contrainte d'analyse "en ligne" d'OLAP car notre algorithme doit pouvoir être arrêté à tout moment afin de ne pas ralentir et perturber l'utilisateur dans son processus d'exploration. Enfin, nous proposons une évaluation utilisateur (voir quelques exemples concernant les matrices dans (Garcia et al., 2010)), car dans la littérature les approches de réorganisation en fouille visuelle sont peu souvent évaluées (par exemple, dans les références suivantes, aucune évaluation utilisateur n'a été proposée (Ankerst et al., 1998) (Choong et al., 2003) (Yang et al., 2003) (Ben Messaoud et al., 2004) (Ben Messaoud et al., 2007) (Sureau et al., 2008)).

3 Algorithme génétique et réorganisation de cubes OLAP

3.1 Représentation et évaluation des individus

Chaque individu de la population représente une réorganisation possible du cube. Plus précisément, on considère un cube ayant dans le cas général trois dimensions $\{D_1, D_2, D_3\}$ dont certaines (ou toutes les trois) sont organisées hiérarchiquement. On note $\{T_1, T_2, T_3\}$ le

triplet d'arbres n-aires correspondant à chacune des trois dimensions du cube. Pour un arbre T_i nous définissons un ensemble de permutations S_i où chaque permutation de S_i est associée à un noeud N de T_i. Une telle permutation locale à N influence l'ordre des fils de N. Un individu est donc représenté par trois ensembles de permutations $\{S_1, S_2, S_3\}$, un pour chaque dimension.

L'évaluation d'un individu se fait à l'aide d'une mesure d'homogénéité "visuelle" qui pour chaque cellule du cube va calculer sa similarité avec son voisinage (voir (Ben Messaoud et al., 2004) (Ben Messaoud et al., 2007)). Cette mesure de similarité est définie par la formule suivante :

$$s(C) = \sum_{x \in \vartheta(C)} |m(C) - m(x)|$$

où C est une cellule du cube, $m(C)$ la valeur de la mesure de la cellule C et $\vartheta(C)$ l'ensemble des cellules voisines C (voisinage de Moore en 3D, soit jusqu'à 26 voisins). Finalement, l'évaluation d'un individu (et donc d'un cube) est égale à :

$$eval(cube) = \sum_{C \in Cube} \frac{s(C)}{max(Cube) - min(Cube)}$$

avec $max(Cube)$ et $min(Cube)$ les valeurs maximum et minimum de la mesure dans le cube. Cette fonction va favoriser les cubes dont les cellules voisines contiennent des valeurs de mesures similaires. Notons que cette fonction n'est pas séparable pour chaque dimension. Si deux mesures sont utilisées, l'évaluation est alors obtenue en calculant la fonction $eval$ pour chaque mesure et en faisant la somme des deux valeurs obtenues.

3.2 Opérateurs de génération aléatoire, de mutation et de croisement

Pour générer la population initiale, nous considérons le cube tel qu'affiché au moment où l'utilisateur déclenche la réorganisation. Un individu représentant ce cube est créé, avec la même organisation que celle affichée. Pour générer aléatoirement de nouveaux individus, il suffit de modifier aléatoirement les permutations des noeuds des arbres T_i. Selon les paramètres choisis par l'utilisateur, l'individu représentant le cube et son organisation initiale peut être présent ou non dans la première génération. En effet, l'utilisateur peut avoir déjà réorganisé partiellement l'arbre et ses choix peuvent ainsi être pris en compte.

La mutation consiste à choisir une permutation au hasard sur une des dimensions et à la modifier avec une probabilité p_{mut}. Deux algorithmes de mutation ont été implémentés, en s'inspirant des travaux qui ont lieu sur la résolution du voyageur de commerce par des algorithmes génétiques (Larranaga et al., 1999) : le "City Swap" qui consiste à choisir aléatoirement deux noeuds d'une même permutation et à les intervertir, et le "2-Opt" qui consiste à choisir aléatoirement deux noeuds d'une permutation et à inverser la séquence contenue entre ces deux noeuds.

A partir de deux individus parents I_1 et I_2, nous avons défini deux opérateurs de croisement (voir (Larranaga et al., 1999)) dans le but de combiner les permutations deux à deux : le croisement uniforme (noté UX) qui pour chaque couple de permutations correspondantes, choisit le noeud courant chez l'un ou l'autre des parents. Si le noeud est déjà présent dans l'enfant, alors il prend le premier noeud suivant chez le parent qui n'est pas chez l'enfant. L'autre croisement

est l'"ordered crossover" (OX), qui pour chaque couple de permutations correspondantes, définit deux points de coupure aléatoirement. Les noeuds se trouvant entre les deux points de coupure sont pris chez l'un des parents, et les autres noeuds sont pris chez l'autre parent. Si lorsque l'on positionne un noeud dans l'enfant, celui-ci est déjà présent dans l'enfant, alors on prend le premier noeud suivant qui n'est pas chez l'enfant.

3.3 Vue d'ensemble de l'algorithme

La sélection utilisée est un tournoi binaire : pour sélectionner un individu, on en choisit deux au hasard et on garde le meilleur. Pour insérer les nouveaux individus dans la population, nous utilisons la méthode "steady state" de Whitley : un seul individu est généré à chaque génération. Il est comparé au pire individu de la population et remplace ce dernier s'il est meilleur. La forme générale de l'algorithme est alors la suivante :

1. Générer la population initiale
2. Générer un individu (méthode "steady state") :
   (a) Sélectionner 2 individus parents I_1 et I_2
   (b) $I \leftarrow Croisement(I_1, I_2)$ avec une probabilité p_{cross}, sinon $I \leftarrow I_1$
   (c) $I \leftarrow Mutation(I)$
   (d) Evaluation de I et insertion éventuelle de I dans la population
3. Si le nombre maximum d'individus générés est atteint alors Stop sinon Retourner en 2

4 Algorithmes de réorganisation "niveau par niveau"

On peut considérer qu'il y a au moins trois manières différentes d'appliquer un algorithme de réorganisation à un cube OLAP : la première possibilité consiste à réorganiser le cube tel qu'il est développé et visualisé par l'utilisateur. C'est la version utilisée par défaut dans notre approche. Si une hiérarchie importante n'est affichée qu'à son premier niveau, alors seulement ce premier niveau sera réorganisé. De fait, cet algorithme utilise des valeurs de mesures agrégées pour les niveaux qui ne sont pas complètement développés. La deuxième possibilité consiste à réorganiser le cube sans tenir compte de la partie affichée et en développant complètement les hiérarchies des dimensions. Ainsi ce sont les hiérarchies complètes des trois dimensions qui sont réorganisées. La troisième possibilité consiste à réorganiser le cube niveau par niveau, en se servant de la possibilité d'agrégation des valeurs dans les hiérarchies OLAP (un noeud de l'arbre porte une valeur agrégée à partir des valeurs contenues dans ses sous-arbres). Pour cela on exécute plusieurs fois l'algorithme de réorganisation. A la première exécution on ne considère que le premier niveau de chaque hiérarchie avec les valeurs agrégées, à la seconde exécution, que le deuxième niveau, et ainsi de suite. Pour l'évaluation des individus, on évalue le cube obtenu en ne tenant compte que du niveau considéré (par exemple, si on réorganise le niveau deux, on évalue le cube obtenu en développant toutes les hiérarchies au niveau deux). Si une dimension du cube ne descend pas au niveau considéré, on utilise pour l'évaluation les feuilles les plus profondes de cette dimension. Cette version est intéressante car elle permet de prendre en compte les valeurs agrégées qui peuvent représenter une information pertinente.

Cube	Dimensions	Mesure(s)	Nb Cellules
Cube 1 (HR)	Time(3), Store(4), Pay Type(1)	Org Salary	$24 \times 25 \times 2 = 1200$
Cube 2 (Sales)	Promotion media(1), Store size(1), Promotions(1)	Unit sales	$14 \times 21 \times 51 = 17136$
Cube 3 (Sales)	Store(2), Promotions(1), Product(2)	Unit sales	$10 \times 51 \times 23 = 11730$
Cube 4 (Sales)	Store Size in SQRFT(1), Time(2), Product(2)	Unit sales Customer count	$21 \times 8 \times 23 = 3864$
Cube 5 (Sales)	Store Size in SQRFT(1), Time(3), Product(2)	Unit sales	$21 \times 24 \times 23 = 11592$

TAB. 1 – *Les différents cubes testés dans l'évaluation. "Dim(d)" indique que pour la dimension "Dim" le niveau considéré dans l'optimisation est "d".*

Nous avons donc défini deux algorithmes de réorganisation "niveau par niveau", cette approche n'ayant pas à notre connaissance été explorée pour OLAP. Le premier algorithme correspond à notre AG : nous utilisons exactement le même algorithme que celui mentionné précédemment, mais en l'appelant plusieurs fois (une fois par niveau présent). Nous notons cet algorithme "Level-GA" dans la suite. Ensuite, nous avons cherché à définir un deuxième algorithme de réorganisation niveau par niveau, mais cette fois en utilisant une heuristique. BEA faisant partie des algorithmes "pionniers" dans ce domaine, nous l'avons donc sélectionné. Cependant, BEA ne peut ni organiser des données hiérarchiques, ni optimiser conjointement les dimensions. Nous avons alors défini un algorithme appelé Level-BEA qui effectue les opérations suivantes : il considère les dimensions une par une. Il optimise la première dimension en commençant par le premier niveau. Cette réorganisation fixe donc l'ordre des noeuds du premier niveau. Ensuite, il développe le cube au niveau 2 et insère les noeuds un par un, mais pas à n'importe quelle position d'insertion comme l'algorithme BEA, mais plutôt en limitant les positions d'insertions à celles fixées par le niveau précédent.

5 Evaluation expérimentale

5.1 Protocole de test

Une série d'expériences a été réalisée. Nous avons tout d'abord cherché les meilleurs paramètres de l'algorithme génétique. Ensuite nous avons comparé notre approche à des algorithmes concurrents. Enfin nous avons réalisé une évaluation utilisateur pour quantifier en pratique l'intérêt de la réorganisation d'un cube OLAP pour des utilisateurs impliqués dans des tâches précises. Pour ces expérimentations, nous avons utilisé les cubes représentés dans la table 1 qui font partie des données de tests "MondrianFoodMart" fournies avec le serveur OLAP Mondrian. Le cube 1 a des cellules bien remplies. Le cube 2 ne possède pas de hiérarchies. Les cubes 3 et 5 sont assez éparses (cellules ayant des valeurs à 0). Le cube 4 sert à tester le cas de deux mesures. Pour le cube 5, à titre indicatif, la taille de l'espace de recherche (nombre de cubes possibles) en tenant compte des hiérarchies est de l'ordre de 10^{47}.

5.2 Recherche du paramétrage de l'algorithme génétique

Nous avons mené de nombreux tests pour parcourir l'espace des paramètres de l'algorithme génétique et choisir un jeu de paramètres adapté pour nos objectifs. Parmi les paramètres considérés, nous avons fait varier les probabilités de croisement et de mutation ($p_{cross} = \{0, 0.5, 1\}$ et $p_{mut} = \{0, 0.1, 0.5, 1\}$), le type de croisement (UX ou OX), le type de mutation ("City Swap" ou "2-opt"), la taille de la population ($\{10, 50, 100, 150, 200, 250\}$). Nous avons utilisé le cube 5 et nous avons limité le nombre d'individus générés à 6000, de manière à ce que l'exécution de l'AG soit de l'ordre de 1 minute. Nous avons estimé qu'un temps de cet ordre de grandeur pouvait être adapté aux contraintes "en ligne" d'OLAP, sachant que dans notre interface (voir section 5.4), l'utilisateur peut continuer à naviguer dans la visualisation pendant l'exécution de l'algorithme de réorganisation. Les tests sont réalisés en moyenne sur 5 essais. Dans les résultats, nous avons observé que le meilleur jeu de paramètres est $p_{cross} = 1$ pour le croisement UX, $p_{mut} = 1$ pour le 2-opt, $|Pop| = 150$. Le croisement UX s'est révélé équivalent ou légèrement supérieur à OX, et la mutation 2-opt est supérieure à city-swap. En ce qui concerne les probabilités de croisement et de mutation, l'utilisation systématique de ces opérateurs donne les meilleurs résultats. Cela s'explique par le fait que l'évolution de type steady-state modifie progressivement la population, et l'équilibre dans l'AG entre intensification et diversification se fait donc grâce à des probabilités élevées pour les opérateurs. En ce qui concerne la taille de la population, les performances sont maximales pour 150 à 200 individus. Au delà de ces valeurs, elles ont tendance à diminuer. Nous avons donc gardé ces paramètres pour tous les autres tests réalisés dans la suite de l'article.

5.3 Etude comparative

Nous avons comparé notre algorithme (noté GA) avec plusieurs autres méthodes possibles. Rappelons que, dans la littérature, aucune méthode de réorganisation pour OLAP ne peut gérer les hiérarchies, et que par conséquent le choix des méthodes concurrentes est restreint. Le premier algorithme concurrent est simplement une génération aléatoire (notée RANDOM), qui permet de connaître le niveau moyen des solutions générées au hasard. Le deuxième algorithme concurrent est un algorithme d'ascension locale ("Hill climbing", noté HC), qui fonctionne de la manière suivante : il part du cube initialn ordonné aléatoirement, puis génère un voisin de ce cube à l'aide de l'opérateur 2-opt. Ce voisin devient l'individu initial s'il est meilleur que celui-ci. Ce type d'approche sert à tester si les principes de l'algorithme génétique sont efficaces ou non par rapport à une méthode stochastique plus simple. Le troisième algorithme concurrent est Level-BEA tel que décrit dans la section 4. Il s'appuie sur une heuristique et sur le principe d'agrégation des niveaux dans OLAP. De manière similaire, nous avons testé l'algorithme génétique travaillant niveau par niveau (noté Level-GA). Enfin, dans l'objectif de réduire les temps d'exécution tout en essayant de garder les meilleures performances, nous avons effectué une hybridation entre Level-BEA et GA. BEA ayant pour principe de partir d'une réorganisation vide, on ne peut donc pas l'utiliser après GA mais plutôt avant. Nous avons donc défini un algorithme hybride noté Level-BEA+GA, dans lequel le cube est optimisé initialement par Level-BEA, et ensuite ce cube est placé dans la population initiale (le reste de la population étant généré aléatoirement). Nous avons alors cherché à diminuer le nombre d'individus générés par l'AG, passant ainsi de 6000 à 3000 afin d'essayer de diviser les temps d'exécution par deux.

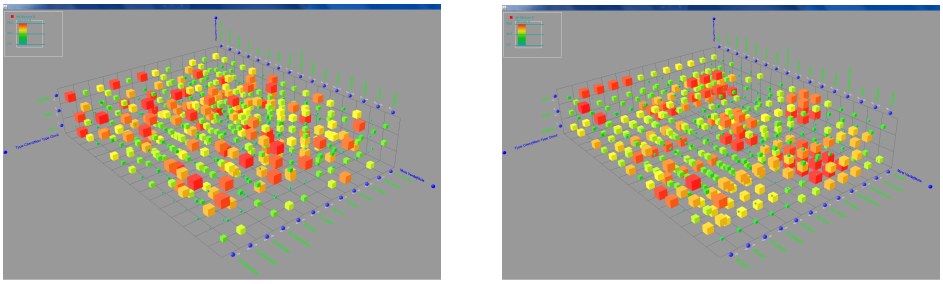
	Cube 1	Cube 2	Cube 3	Cube 4	Cube 5
GA	1757,4±10,7 17,6s±0,1	184,6±0,7 54,2s±0,9	408,0±4,4 59,2s±0,3	757,3 ±34 34,8 s ±1,6	97,9±2,1 68,5s±1,9
RANDOM	1834,7± 11,2 17,2s±1,8	197,6±0,9 54,5s±0,1	472,6±6,2 57,9s±0,2	1369,8 ±76,8 32,8 s ±0,1	194,8±6,3 66,7s±0,5
HC	1848,8±62,3 15,7s±0,0	188,5±2,4 53,6s±1,4	465,0±91,2 58,8s±0,2	779,5 ±28,1 32,9 s ±0,2	118,5±8,2 67,2s±0,5
Level-BEA	1908,2±53,6 0,2s±0,0	358,0±12,7 1,4s±0,2	707,7±61,7 3,8s±0,0	925,4 ±63,3 1,2 s ±0,1	130,7±4,4 1,7s±0,0
Level-GA	1816,8±1,7 32,0s±0,1	184,0±0,6 56,7s±0,7	409,4±4,1 46,2s±0,3	826,1 ±34,4 33,3 s ±0,3	106,7±5,4 73,6s±3,8
Level-BEA +GA	1768,0±12,4 9,5s±0,0	186,6±0,9 30,7s±2,4	428,2±5,4 35,2s±0,4	796,6 ±42,8 19,6 s ±0,1	111±2,5 37,2s±0,9

TAB. 2 – *Résultats obtenus par les différents algorithmes sur les cubes mentionnés dans la table 1, en moyenne sur 5 essais. Pour une méthode donnée, la première ligne donne la valeur de coût obtenue, et la deuxième ligne donne le temps d'exécution total.*

Les résultats sont présentés dans la table 2. On constate que l'approche génétique (GA) fait partie systématiquement des algorithmes donnant les meilleurs résultats. Ses performances semblent être fiables par rapport aux autres algorithmes qui, pour certains cubes, obtiennent de bonnes performances, mais aussi de mauvaises pour d'autres cubes. En ce qui concerne l'optimisation niveau par niveau, on peut remarquer que cette approche n'est pas aussi pertinente que nous le pensions. En effet, utiliser les valeurs agrégées dans l'arbre ne semble pas guider les deux algorithmes vers les meilleures solutions. En ce qui concerne l'hybridation Level-BEA+GA n'obtient pas toujours les meilleures performances, mais on peut constater que le temps d'exécution a été nettement diminué. Ainsi, l'hybridation semble une piste intéressante à poursuivre, surtout si l'on trouve un moyen de fusionner plus BEA et GA en intégrant BEA comme une forme de mutation par exemple dans l'algorithme génétique. En terme de temps d'exécution, Level-BEA est de loin le plus rapide. Cependant il n'est pas très fiable car ses performances sont trop inégales d'un cube à l'autre, un point qui se confirme visuellement par l'aspect des cubes obtenus (qui ne sont clairement pas proches de l'optimum). Toutefois, si le critère de vitesse devient crucial, Level-BEA peut être intéressant afin de fournir un résultat approximatif à l'utilisateur.

5.4 Interface réalisée et évaluation utilisateur

Dans nos travaux sur OLAP, nous étudions plus généralement des interfaces visuelles et interactives. Nous avons ainsi développé un prototype, VR4OLAP, qui représente un cube OLAP comme un cube 3D (voir figure 1) dans lequel trois dimensions hiérarchiques peuvent être représentées ainsi qu'une ou deux valeurs de mesures. Cette visualisation est interactive et de nombreux opérateurs OLAP y ont été intégré sous la forme d'objets graphiques cliquables. En cliquant sur ces objets, l'utilisateur déclenche les opérateurs OLAP. En ce qui concerne la réorganisation, l'utilisateur déclenche son exécution de manière incrémentale, par pas durant



(a) (b)

FIG. 1 – *Illustration de notre interface et de la réorganisation d'un cube (entre (a) et (b)).*

moins de 1 minute (pour un cube ayant environ 10000 cellules). L'utilisateur peut donc cliquer une première fois pour obtenir un résultat et donc un nouveau cube, et s'il souhaite attendre plus longtemps pour améliorer ce résultat, il peut cliquer à nouveau. Ainsi, l'utilisation de l'algorithme génétique ne vient pas rallonger outre mesure l'utilisateur dans son exploration du cube. De plus, pendant l'exécution de la réorganisation, l'utilisateur peut continuer à naviguer dans le cube.

Nous avons cherché à savoir si l'amélioration mesurée dans les résultats précédents grâce à la fonction d'homogénéité correspondait également à une amélioration de la perception des données par l'utilisateur, ou à l'amélioration des performances de l'utilisateur dans certaines tâches. Pour cela, nous avons défini trois tâches à résoudre par les utilisateurs :

- T1 : cette tâche consiste à trouver dans une dimension du cube deux membres ayant le même comportement vis à vis de la mesure (i.e. 2 "tranches" identiques du cube). Pour cela nous avons défini des données OLAP en nous inspirant de la base MondrianFoodMart fournie avec le serveur OLAP Mondrian que nous utilisons dans VR4OLAP,
- T2 : cette tâche consiste à trouver dans un cube un membre d'une dimension plus atypique que les autres. Nous avons défini des classes de membres (comportements similaires pour la mesure), et le membre à trouver est celui dont le comportement n'appartient à aucune des classes définies,
- T3 : pour cette tâche il s'agit de trouver le nombre de classes des membres d'une dimension. Nous avons défini de manière similaire aux précédentes tâches des données dans lesquelles les membres appartiennent à des classes.

Deux méthodes sont testées. La première affiche le cube sans ordre particulier sur les dimensions : les arbres sont triés dans l'ordre alphanumérique. La deuxième consiste à utiliser l'algorithme génétique pour réorganiser le cube. Pour chaque tâche et pour chaque méthode testée, la réponse donnée par l'utilisateur est enregistrée pour pouvoir plus tard la comparer avec la valeur attendue et ainsi mesurer la qualité de la réponse. Lorsque l'utilisateur ne répond pas, la qualité est de 0. Si la réponse est juste, la qualité vaut 1. Pour une réponse intermédiaire, nous mesurons sa qualité en fonction de sa similarité avec la réponse exacte. De plus, pour chacune des questions, le temps de réponse a été mesuré. Un questionnaire préalable permet de connaître le niveau de la personne en ce qui concerne OLAP. Un questionnaire final permet de connaître les impressions "à chaud" de l'utilisateur. Avant de répondre aux questions, nous laissons l'utilisateur interagir avec les visualisations dans le but qu'il se familiarise avec elles

Tâche	Méthode	Temps (s)	Qualité	Nb rép. exactes	Nb d'abandons
T1	sans réorg.	128 ± 89	0.67 ± 0.47	4	2
T1	avec réorg.	39 ± 33	$0.95 \pm 0{,}11$	6	0
T2	sans réorg.	61 ± 67	$0.75 \pm 0{,}41$	5	1
T2	avec réorg.	25 ± 15	1.00 ± 0.0	7	0
T3	sans réorg.	63 ± 29	0.91 ± 0.15	5	0
T3	avec réorg.	29 ± 23	0.91 ± 0.15	5	0

TAB. 3 – *Résultats obtenus lors de l'évaluation utilisateur pour les différentes tâches (T_1 à T_3), avec ou sans réorganisation (dans ce dernier cas, l'ordre choisi pour les membres de chaque niveau est l'ordre alphabétique).*

et obtienne les compétences minimales nécessaires pour répondre aux questions posées. Sept utilisateurs ayant déjà des connaissances en OLAP ont été recrutés (niveau bac+2 à bac+5).

Les résultats sont présentés dans la table 3. Nous pouvons remarquer que l'amélioration des performances des utilisateurs est visible avec l'aide de la réorganisation, à la fois en terme de qualité et en terme de temps. L'amélioration mesurable sur la fonction d'homogénéïté correspond donc bien à une amélioration visuelle des cubes. Nous pensons donc que l'attente de 1 minute environ pour l'utilisateur est justifiée au regard de l'amélioration produite en terme de qualité, de nombre d'abandons, et de temps. Cette amélioration est bien sur liée aux tâches fixées et qui ont concerné la découverte de groupes ou de données aberrantes. On ne peut pas la généraliser à d'autres types de tâches sans faire de tests complémentaires. Notre étude confirme pour la 3D d'autres études réalisées en 2D (voir survol dans (Garcia et al., 2010)) où les utilisateurs ont apprécié la réorganisation automatique de visualisations matricielles.

6 Conclusions et perspectives

Nous avons étudié dans cet article la réorganisation de cubes OLAP dans le but d'améliorer les représentations visuelles de ces cubes et au final les performances de l'utilisateur. Nous avons proposé un nouvel algorithme génétique pour la réorganisation d'arbres n-aires pouvant optimiser de manière conjointe les dimensions hiérarchiques d'un cube de données. Nous avons étudié le paramétrage de cet algorithme afin d'obtenir un rapport performance/temps qui soit acceptable dans le cadre d'une analyse en ligne. Nous avons proposé un algorithme heuristique inspiré de BEA afin d'organiser niveau par niveau les dimensions hiérarchiques. Nous avons étendu l'algorithme génétique de la même manière. Lors de tests comparatifs, nous avons montré que notre approche génétique est compétitive par rapport à des algorithmes du domaine, même en limitant le temps d'exécution à une minute. De plus, notre approche traite les dimensions hiérarchiques ce qui n'avait encore jamais été réalisé dans le cadre des visualisations de cubes de données. Nous avons intégré les différents algorithmes dans une interface 3D pour OLAP, et nous avons réalisé une évaluation utilisateur qui a montré l'efficacité de la réorganisation dans des tâches impliquant la détection de similarités entre données.

Nous pensons pouvoir améliorer l'algorithme génétique selon plusieurs directions. Tout d'abord nous souhaitons effectuer une hybridation plus rapprochée avec Level-BEA que celle

que nous avons testée. Plus précisément, nous étudions comment intégrer BEA comme un opérateur de mutation heuristique dans l'algorithme génétique. Egalement, un des intérêts des AG est leur capacité à être facilement parallélisés. Nous allons donc étudier comment la fonction d'évaluation, qui est couteuse en temps de calcul, peut être implémentée sur une architecture parallèle de type GPU. Notre algorithme pourrait alors obtenir des temps de calculs comparables à ceux de Level-BEA, avec une qualité des résultats nettement améliorée. Une autre interrogation que nous avons porte sur une comparaison avec une approche optimale utilisant la programmation dynamique, comme dans (Bar-joseph et al., 2003). Certes la complexité de cette méthode est importante, et ce type d'approche n'optimise pas les dimensions de manière conjointe, mais cependant la taille des cubes affichés dans OLAP reste peut être suffisamment petite pour qu'une telle méthode puisse avoir un intérêt. Enfin, nous souhaiterions mieux évaluer l'intérêt d'optimiser les dimensions de manière conjointe par rapport à une optimisation séparée. L'optimisation conjointe est traitée par très peu de méthodes, alors que nous avons montré ici que l'optimisation séparée (comme dans Level-BEA) tient mal compte des dépendances pouvant exister entre lignes et colonnes. Dans cette problématique, l'aspect utilisateur rentre aussi en jeu, et nous aimerions savoir si une différence entre optimisation conjointe et optimisation séparée peut être perçue visuellement de manière significative.

Références

Ankerst, M., S. Berchtold, et D. A. Keim (1998). Similarity Clustering of Dimensions for an Enhanced Visualization of Multidimensional Data. Volume 0, Los Alamitos, CA, USA, pp. 52. IEEE Computer Society.

Bar-joseph, Z., E. D. Demaine, D. K. Gifford, M. Hamel, et T. S. Jaakkola (2003). K -ary clustering with optimal leaf ordering for gene expression data. *Bioinformatics 19*(9), 1070–1078.

Bar-joseph, Z., D. K. Gifford, et T. S. Jaakkola (2001). Fast Optimal Leaf Ordering For Hierarchical Clustering. *Bioinformatics 17*, 22–29.

Barbará, D. et M. Sullivan (1997). Quasi-cubes : exploiting approximations in multidimensional databases. *SIGMOD Rec. 26*(3), 12–17.

Ben Messaoud, R., O. Boussaid, et S. Loudcher Rabaséda (2007). A Multiple Correspondence Analysis to Organize Data Cubes. In *Proceeding of the 2007 conference on Databases and Information Systems IV : Selected Papers from the Seventh International Baltic Conference*, pp. 133–146.

Ben Messaoud, R., S. Loudcher, O. Boussaid, et F. Bentayeb (2004). OpAC : A New OLAP Operator Based on a Data Mining Method. In *Sixth International Baltic Conference on Databases and Information Systems*, pp. 417–420.

Bertin, J. (1981). *Graphics and graphic information-processing / Jacques Bertin ; translated by William J. Berg and Paul Scott*. de Gruyter, Berlin ; New York :.

Caraux, G. (1984). Réorganisation et représentation visuelle d'une matrice de données numériques : un algorithme itératif. *Revue de statistique appliquée 32*, 5–23.

Chaudhuri, Q. et U. Dayal (1997). An overview of data warehousing and OLAP technology. *ACM SIGMOD Record 26*, 65–74.

Choong, Y. W., D. Laurent, et P. Marcel (2003). Computing appropriate representations for multidimensional data. *Data Knowl. Eng. 45*(2), 181–203.

Climer, S. et W. Zhang (2006). Rearrangement Clustering : Pitfalls, Remedies, and Applications. *The Journal of Machine Learning Research 7*, 919–943.

Codd, E. F., S. B. Codd, et C. T. Salley (1993). Providing OLAP to User-Analysts : An IT Mandate. Technical report, E.F. Codd and Associates.

Cuzzocrea, A. et S. Mansmann (2009). OLAP visualization : models, issues, and techniques. *Encyclopedia of Data Warehousing and Mining, 2nd ed.*, 1439–1446.

Fekete, J.-D. et N. Henry (2009). Matrix reordering survey. In *Visualisation Summer School, Peking University*.

Garcia, M., W. Huang, C. Seifert, et W. Wallisch (2010). Literature Survey : The Reorderable Matrix.

Larranaga, P., C. M. H. Kuijpers, R. H. Murga, I. Inza, et S. Dizdarevic (1999). Genetic Algorithms for the Travelling Salesman Problem : A Review of Representations and Operators. *Artificial Intelligence Review 13*, 129–170.

Liiv, I. (2010). Seriation and matrix reordering methods : An historical overview. *Statistical Analysis and Data Mining 3*(2), 70–91.

McCormick, W. T., P. J. Schweitzer, et T. W. White (1972). Problem Decomposition and Data Reorganization by a Clustering Technique. *Operations Research 20*, 993–1009.

Morris, S. A., B. Asnake, et G. G. Yen (2003). Optimal dendrogram seriation using simulated annealing. *Information Visualization 2*(2), 95–104.

Sureau, F., F. Bouali, et G. Venturini (2008). On improving OLAP visualizations with rearrangement clustering. In *SFC-CLADAG 08 : first joint meeting of the Sociï£¡tï£¡ Francophone de Classification and the Classification and Data Analysis Group of the Italian Statistical Society*, pp. 417–420.

Vandev, D. L. et Y. G. Tsvetanova (1997). Ordering of Hierarchical Classifications.

Yang, J., W. Peng, M. Ward, et E. Rundensteiner (2003). Interactive hierarchical dimension ordering, spacing and filtering for exploration of high dimensional datasets. *Proc. IEEE Symposium on Information Visualization.*, 105–112.

Summary

In this paper we propose a new algorithm for the hierarchical reorganization of OLAP cubes, with the aim of improving their visualization. This algorithm is characterized by the fact that it deals with hierarchically organised dimensions and by the fact that it can optimize dimensions in conjunction. It uses a genetic algorithm that reorganizes n-ary trees. It has been integrated in an OLAP interface and has been tested in comparison with other rearrangement clustering approaches. It obtains significantly better results. From this point of view, we have also proposed a hierarchical version of the BEA algorithms. Finally, our approach has been evaluated with real users and the results highlight the interest of reorganization in several OLAP tasks.