# Structure Réutilisable pour le Calcul et la Manipulation des Cubes de Données

Hassani Hachim, Noël Novelli

LIF-CNRS UMR 6166 – Case 901
Université de la Méditerranée
Faculté des Sciences de Luminy ; 163, Avenue de Luminy
13288, Marseille Cedex 9, FRANCE
{hachim, novelli}@lif.univ-mrs.fr

**Résumé.** Les cubes de données sont de plus en plus utilisés pour le pré-calcul de requêtes OLAP afin de permettre essentiellement à des analystes de trouver des tendances ou des anomalies dans de grandes quantités de données. Il se révèle que tout problème lié aux cubes de données est coûteux, que ce soit pour la construction, la matérialisation, la manipulation ou la mise à jour. Dans cet article, nous introduisons la notion de pré-calcul de cubes de données et la caractérisation associée qui est basée sur le modèle partitionnel. A notre connaissance, aucune des approches actuelles ne s'est intéressée à la réutilisation de pré-calcul des cubes de données. Pourtant cette dernière permet de calculer et de manipuler efficacement des cubes de données dans plusieurs contextes comme les applications météorologiques, le calcul de requêtes à la volée ou encore le calcul de plusieurs cubes de données en réseau ou en local.

## 1 Introduction

Conceptuellement, l'opérateur GROUP BY CUBE introduit dans Gray et al. (1996) représente toutes les combinaisons des valeurs des dimensions auxquelles sont associées des mesures (valeurs numériques). C'est un outil analytique inestimable pour les applications nécessitant une analyse de quantités volumineuses de données comme on en trouve dans les systèmes de support de décision, le *business intelligence* et le datamining. De telles applications nécessitent des réponses rapides aux requêtes, ad-hoc pour la plupart, sur une base de données ou un entrepôt de données. Le cube de données constitue l'une des représentations possibles permettant d'optimiser ce type de requêtes car il correspond au pré-calcul de toutes les requêtes possibles pour une fonction agrégative donnée.
Cet opérateur exécute les GROUP BY correspondant à toutes les combinaisons possibles de l'ensemble de dimensions. La figure 1 montre l'équivalence sémantique entre l'opérateur GROUP BY CUBE et l'union de toutes les combinaisons de GROUP BY pour la table $UneTable$ avec $Dim = \{A, B, C\}$ l'ensemble des dimensions, $\mathcal{M}$ la mesure et $f$ la fonction agrégative. Cette requête représente $2^{|Dim|}$ GROUP BY et chaque résultat de GROUP BY est un cuboïde.

```
SELECT A, B, C, f(M) FROM UneTable GROUP BY CUBE( A, B, C )
                          <=>
SELECT A,    B,    C,    f(M) FROM UneTable GROUP BY A, B, C UNION
SELECT A,    B,    'ALL', f(M) FROM UneTable GROUP BY A, B    UNION
SELECT A,    'ALL', C,    f(M) FROM UneTable GROUP BY A,    C UNION
SELECT 'ALL', B,    C,    f(M) FROM UneTable GROUP BY    B, C UNION
SELECT A,    'ALL', 'ALL', f(M) FROM UneTable GROUP BY A       UNION
SELECT 'ALL', B,    'ALL', f(M) FROM UneTable GROUP BY    B    UNION
SELECT 'ALL', 'ALL', C,    f(M) FROM UneTable GROUP BY       C UNION
SELECT 'ALL', 'ALL', 'ALL', f(M) FROM UneTable
```

**FIG. 1** – *Équivalence SQL entre l'opérateur GROUP BY CUBE et l'union de tous les GROUP BY pour la relation* $UneTable$ *(*$Dim = \{A, B, C\}$*)*

Après l'introduction de l'opérateur GROUP BY CUBE, de nombreux chercheurs se sont intéressés à la complexité du problème. Il se révèle que tout problème associé aux cubes de données est difficile, que ce soit pour la construction, la matérialisation, la manipulation ou la mise à jour. En effet, l'espace de recherche d'un cube de données est l'ensemble des parties de l'ensemble des dimensions donc de taille exponentielle.

Notre objectif principal est d'introduire une approche de pré-calcul de cubes afin d'économiser les calculs communs pour les problématiques liées à la construction, la matérialisation et la manipulation des cubes de données pour toute fonction agrégative. Pour montrer l'efficacité de notre pré-calcul et la structure associée, nous avons exploité sa réutilisation dans les quatre points innovants suivants :

1. **Recalcul de cubes de données** sur des relations où les combinaisons de valeurs de dimensions ne changent pas (données météorologiques, capteurs de données...).
   Ce type d'application est une utilisation directe de notre pré-calcul. On peut rencontrer ce type de cas sur des données provenant de capteurs (données météorologiques, de pollution...) où seules les valeurs de mesures (valeurs mesurées par les capteurs) changent au fil du temps, mais pas les valeurs des dimensions (position des capteurs...). Nous construisons la structure une seule fois (basée sur les combinaisons et les valeurs des dimensions) indépendamment des valeurs de mesure de la relation initiale. Pour chaque période où les valeurs de mesure changent, nous pouvons calculer rapidement (comme le montre notre expérimentation) le nouveau cube de données pour toute fonction agrégative.
2. **Calcul de plusieurs cubes de données** : programmation parallèle et distribuée.
   Ceci permet d'améliorer considérablement le calcul de plusieurs cubes de données sur une même relation c'est-à-dire plusieurs fonctions agrégatives. A notre connaissance, aucune des approches actuelles ne propose de méthode efficace pour ce calcul. Elles ne prennent en compte qu'une seule fonction agrégative à la fois lors du calcul d'un cube de données. Si l'analyste souhaite appliquer une autre fonction, tout le processus de calcul est repris sans pouvoir bénéficier des calculs précédents. Le pré-calcul du cube de données nous permet de résoudre ce problème et d'économiser beaucoup de temps de calcul. De plus, notre méthode fonctionne aussi bien sur une seule machine que sur un réseau de plusieurs machines.

3. **Calcul de Cubes de Données partiels**
   Ce point de notre contribution concerne principalement des très grandes relations. Pour des relations très grandes, il est difficile voire impossible de calculer ou stocker les cubes de données. Dans ce cas nous proposons une technique permettant de calculer et stocker seulement la partie du cube qui rentre en mémoire. Le reste du cube pourra être calculé à tout moment à partir des fragments de cubes déjà calculés. La différence entre notre approche et les approches proposant des algorithmes de sélection de vues à matérialiser, est que nous pouvons calculer n'importe quel cuboïde avec n'importe quelle fonction agrégative à tout moment contrairement à ces approches qui ne prennent en compte qu'une seule fonction agrégative spécifiée préalablement par l'analyste.
4. **Calcul de cubes de données à la volée** : Navigation interactive dans plusieurs cubes de données.
   Le dernier point rejoint le troisième, mais dans un contexte différent et totalement nouveau pour la manipulation de cubes de données : il permet d'une part de réduire les besoins mémoire lors d'une analyse de données suivant plusieurs fonctions agrégatives, mais surtout d'offrir une très grande facilité de navigation et interaction avec les cubes de données. L'analyste peut choisir à la volée les cuboïdes à calculer, avec une ou plusieurs fonctions agrégatives en même temps. Les besoins mémoire sont donc réduits puisqu'au lieu de calculer toutes les valeurs et de les conserver en mémoire, nous ne calculons que celles qui sont potentiellement intéressantes pour l'analyse. De plus, nos expérimentations montrent que le temps de calculs à la volée est très rapide.

**Organisation du papier**
Après le survol des concepts liés aux cubes de données dans cette section, nous présentons quelques travaux significatifs dans la section 2 menés sur la construction des cubes de données. Dans la section 3 nous introduisons le pré-calcul de cubes de données et la caractérisation de la structure réutilisable associée, puis nous montrons comment réutiliser cette structure. Enfin, nous montrons les résultats de nos expérimentations dans la section 4, avant de conclure et discuter des perspectives de ce travail dans la section 5.

# 2 Etat de l'art

L'introduction de la notion de cube de données dans Gray et al. (1996) a engendré de très nombreuses publications. Le problème général s'avère NP-Complet (Karloff et Mihail (1999)) et sa complexité est exponentielle sur presque chaque aspect.
Les travaux proposés sont orientés vers deux grandes catégories. La première consiste à améliorer les temps de calculs de cubes de données, et la seconde à réduire les besoins en mémoire de stockage. Parmi les premières approches proposées, on trouve des approches d'optimisation basées sur des techniques de tri ou de hachage (Gray et al. (1996); Beyer et Ramakrishnan (1999); Eisenberg et Melton (2000); Colossi et al. (2002)). L'objectif de ces approches est double : optimiser le tri lui-même et diminuer le nombre de tris redondants (recouvrement de tris, mise en pipeline des tris...). D'autres approches ont utilisé des techniques de représentation multidimensionnelle sous forme de tableaux comme dans Zhao et al. (1997) avec une indexation directe évitant les tris. Ces approches sont très efficaces si le nombre de dimensions est petit. En effet, la représentation multidimensionnelle requière beaucoup de mémoire et est

donc mal adaptée à la fois aux grands nombres de dimensions et aux données éparpillées. Afin de palier l'éparpillement des données, des approches comme Ross et Srivastava (1997); Beyer et Ramakrishnan (1999) ont été introduites et exploitent cet éparpillement notamment en effectuant un pré-traitement sur les données afin de gommer l'éparpillement.
Comme la taille des cubes de données est exponentielle au nombre de dimension de la relation initiale, les temps d'écriture sur disque représentent une durée non négligeable par rapport aux temps de calculs. Partant de ce constat, des contributions se sont intéressées à des représentations plus "petites" des cubes de données en utilisant des arbres de préfixes, des représentations compactes ou condensées, des résumés avec ou sans perte d'information. Plusieurs techniques d'indexation ont donc été conçues pour stocker des cubes de données d'une manière efficace. Forest Cube Johnson et Shasha (1997), exploite la redondance de préfixe pour stocker le cube, les préfixes uniques sont stockés une seule fois, mais l'arbre contient tous les chemins possibles (même les chemins inexistants) le rendant inadéquat surtout pour des données éparpillées. Cette structure a été par ailleurs aussi utilisée en datamining pour la recherche de motifs fréquents, FP-Tree Han et al. (2000).
La notion de cubes condensés a été introduite et plusieurs approches exploitent les redondances inhérentes au cube pour en réduire la taille. Ces approches cherchent donc à minimiser les temps de calculs et/ou les temps d'écriture du cube en réduisant la taille du cube (Ng et Ravishankar (1997); Cicchetti et al. (2001); Lakshmanan et al. (2002); Wang et al. (2002); Lakshmanan et al. (2003a,b); Casali et al. (2006)). D'autres approches proposent des techniques pour réduire considérablement le besoin de stockage en construisant des résumés de cubes (Goil et Choudhary (1998); Sismanis et Roussopoulos (2003); Casali et al. (2003b)) ou en approximant le cube de données (Vitter et al. (1998); Shanmugasundaram et al. (1999); Barbará et Wu (2000); Saint-Paul et al. (2005); Jermaine et al. (2005)). Certaines de ces approches se trouvent dans les deux catégories ce qui les rend particulièrement efficaces.
La construction parallèle de cubes de données a fait l'objet de plusieurs études pour améliorer le temps de calcul et diminuer les mouvements de données. Les approches actuelles de ce domaine peuvent être classées en deux catégories, le partage du travail (Lu et al. (1997); Dehne et al. (2001b,a)) et le partage des données (Goil et Choudhary (1998, 1999); Chen et al. (2004)). Dans notre approche, nous adoptons une stratégie mixte qui consiste à partager le travail et les données entre plusieurs unités de calcul.

# 3 Caractérisation de notre structure réutilisable : DVCube

La caractérisation que nous introduisons dans ce paragraphe est un pré-calcul de cubes de données, calculable sur un ordinateur ou sur plusieurs en réseau. Cette caractérisation est mise en œuvre dans l'algorithme que nous décrivons dans le paragraphe 3.2. En outre, notre caractérisation permet de bénéficier des avantages des deux familles d'approches de calcul parallèle ou distribué de cubes de données (partage du travail et partage des données). En effet, elle nous permet de partager le travail sur plusieurs machines et de partager les données sur chacune sans aucune redondance.
La caractérisation de notre structure est basée sur la notion de partition introduite dans Cosmadakis et al. (1986); Spyratos (1987), utilisée dans TANE (Huhtala et al. (1999)), Dep-Miner (Lopes et al. (2000)) et FUN (Novelli et Cicchetti (2001b,a)) pour l'extraction de dépendances

fonctionnelles à partir de relations existantes et plus récemment pour le calcul efficace des cubes de données dans Cicchetti et al. (2001); Li et al. (2004); Casali et al. (2006).

## 3.1 Notation et définitions

Pour illustrer les notations et définitions que nous introduisons dans cette section, nous utilisons une relation exemple *Car Sales* (cf. Table 1) qui représente des ventes de voitures. Les dimensions d'analyse sont $Seller$ (qui fait la vente), $Category$ (de voiture), $City$ (lieu de vente) et $Customer$ (l'age de l'heureux propriétaire). Une seule mesure est présente (valeur qui représente le nombre de ventes de voitures).
Dans la suite de cet article, nous considérons la relation *Car Sales* ci-dessus dans nos exemples.

| *Car Sales* | | | | | |
|---|---|---|---|---|---|
| *IdRow* | *Seller* | *Category* | *City* | *Customer* | *Value* |
| *1* | **J**enny | **C**ity cars | **M**iami | **Y**oung | 10 |
| *2* | **J**enny | **S**port cars | **M**iami | **A**dult | 20 |
| *3* | **E**lodie | **S**port cars | **M**iami | **Y**oung | 30 |

TAB. 1 – *Relation exemple* Car Sales *représentant des ventes de voitures*

Inspiré du concept de partition, nous introduisons le concept de Dimensional-Value Partition sur un ensemble de dimensions $X \subseteq Dim$ (avec $Dim$ l'ensemble des dimensions de la relation initiale). Une telle définition nécessite d'étendre le concept de classe d'équivalence de tuples que nous appelons dans notre approche Dimensional-Value Equivalence Class (notée DV-Class).

Nous désignons par $r$ une relation de schéma $R = \{IdRow, Dim, \mathcal{M}\}$ et $X \subseteq Dim$. Les dimensions de $Dim$ sont totalement ordonnées, $Dim = \{d_1, ..., d_{|Dim|}\}$. La valeur symbolique $ALL$, notée aussi $'?'$, généralise toutes les valeurs possibles de toutes les dimensions i.e. $\forall d_i \in Dim, \forall a \in r[d_i], \{a\} \subset ALL$ ce qui signife que tout élément de toute dimension est un sous ensemble de la valeur symbolique $ALL$.
De plus :

- $t[X]$ représente la projection du tuple $t \in r$ sur $X \subseteq Dim$.
  Illustration : $t_1[Seller] = J$
- $t < X >=< x_1, ..., x_{|Dim|} >$(Casali et al. (2003a)) est un n-uplets à $|Dim|$ éléments tel que
  $$\forall i \in [1..|Dim|], x_i = \begin{cases} t[d_i] \text{ si } d_i \in X \\ ALL(?) \text{ sinon} \end{cases}$$
  Illustration : $t_1 < Seller >= J??? = t_2 < Seller >$
- $S_t[X]$ est un ensemble d'identifiants de tuples de la relation $r$ partageant la même valeur $t[X]$ tel que $S_t[X] = \{u[IdRow] \mid u[X] = t[X], \forall u \in r\}$ (Cosmadakis et al. (1986); Spyratos (1987)).
  Illustration : $S_{t_1}[Seller] = \{1, 2\} = S_{t_2}[Seller]$ (les tuples 1 et 2 partagent la même valeur $J$ pour la dimension $Seller$).

**Définition 1** *Dimensional-Value Class (DV-Class)*
*Soit une relation $r$ de schéma $R = \{IdRow, Dim, \mathcal{M}\}$ et $X \subseteq Dim$. La DV-Class du tuple*

*$t \in r$ selon $X$, noté $[t]X$, est définie comme suit :* $[t]X =< S_t[X], t < X >>$.
*On note $[\emptyset]X$ la DV-Class dont $S_t[X] = \emptyset$.*

**Exemple 1** *Prenons $S_{t_1}[Seller] = \{1, 2\}$ et $t_1 < Seller >= J???$.*
*On obtient* $[t_1]Seller =< \{1, 2\}, J??? >$

**Définition 2** *: Opérateur Somme*
*Soit une relation $r$ de schéma $R = \{IdRow, Dim, \mathcal{M}\}$, $X$ et $Y \subseteq Dim$ tel que $X \cap Y = \emptyset$ et, $t$ et $t' \in r$.*
*L'opérateur somme de deux n-uplets $t < X >$ et $t' < Y >$, noté $\oplus$, est définie comme suit :*
$t < X > \oplus t' < Y >= \begin{cases} t[d_i] \text{ si } t[d_i] \neq ALL \\ t'[d_i] \text{ sinon} \end{cases}$ , $\forall i \in [1..|Dim|]$

**Exemple 2** *La somme des deux n-uplets $t_1 < Seller >= J???$ et $t_2 < Category >=?S??$ est $t_1 < Seller > \oplus t_2 < Category >= JS??$*

**Définition 3** *: Intersection de deux DV-Class*
*Soit une relation $r$ de schéma $R = \{IdRow, Dim, \mathcal{M}\}$, $X$ et $Y \subseteq Dim$ tel que $X \cap Y = \emptyset$ et, $t$ et $t' \in r$.*
*Soient $[t]X =< S_t[X], t < X >>$ et $[t']Y =< S_{t'}[Y], t' < Y >>$ deux DV-class, l'intersection de $[t]X$ avec $[t']Y$, notée $[t]X \cap [t']Y$, est définie comme suit :*
$[t]X \cap [t']Y = [t'']\{Z\}$ *où* $Z = \{X \cup Y\}$ *et* $t'' \in r$ *tel que* $t''[IdRow] \in \{S_t[X] \cap S_{t'}[Y]\}$.
$[t'']\{Z\} =< S_{t''}[Z], t'' < Z >>$ *avec* $\begin{cases} S_{t''}[Z] = \{S_t[X] \cap S_{t'}[Y]\} \text{ et} \\ t'' < Z >= t < X > \oplus t' < Y > \end{cases}$
*Notons que d'après la définition 1, si $\{S_t[X] \cap S_{t'}[Y]\} = \{\emptyset\}$ alors $t'' = [\emptyset]$.*

Remarques :
Pour une relation à $n$ tuple, $[t]\emptyset = \{< \{1, ..., n\}, ?...? >\}$.
De plus, $[t]\emptyset \cap [t]X = [t]X \cap [t]\emptyset = [t]X$.

**Exemple 3** *L'intersection des deux DV-Class $[t_1]Seller =< \{1, 2\}, J??? >$ et $[t_1]Category =< \{1\}, ?C?? >$ est $[t_1]Seller \cap [t_1]Category =< \{1, 2\} \cap \{1\}, J??? \oplus ?C?? >=< \{1\}, JC?? >$.*

**Définition 4** *Dimensional-Value Partition (DV-Partition)*
*Soit une relation $r$ de schéma $R = \{IdRows, Dim, \mathcal{M}\}$ et $X \subseteq Dim$. La DV-Partition de $r$ suivant $X$, notée $\Pi_X(r)$ est définie comme suit :* $\Pi_X(r) = \bigcup_{X' \subseteq X} [t]X', \forall t \in r$.

**Exemple 4** *La DV-Partition $\Pi_{Seller}(r)$ est l'union de toutes les DV-Class selon la dimension $Seller$.*
$[t]\emptyset =< \{1, 2, 3\}, ???? >$
$[t_1]Seller = [t_2]Seller =< \{1, 2\}, J??? >$
$[t_3]Seller =< \{3\}, E??? >$
*d'où* $\Pi_{Seller}(r) = \{< \{1, 2, 3\}, ???? >, < \{1, 2\}, J??? >, < \{3\}, E??? >\}$.

Remarques :
Pour une relation $r$ de $n$ tuples, $\Pi_\emptyset(r) = [t]\emptyset = \{< \{1, 2, ..., n\}, ?...? >\}$.
Le DVCube de la relation est la DV-Partition de $r$ suivant $Dim$ : $\Pi_{Dim}(r)$.

**Définition 5** *Produit de DV-Partitions*
*Soit une relation* $r$, $\Pi_X(r)$ *et* $\Pi_Y(r)$ *deux DV-Partitions de* $r$ *suivant* $X$ *et* $Y$. *Le produit de deux DV-Partitions* $\Pi_X(r)$ *et* $\Pi_Y(r)$, *noté* $\Pi_X(r) \bullet_{dv} \Pi_Y(r)$ *est défini comme suit :*
$\Pi_X(r) \bullet_{dv} \Pi_Y(r) = \{z | z = x \cap y \neq [\emptyset]\{X \cup Y\}$ *avec* $x \in \Pi_X(r)$ *et* $y \in \Pi_Y(r)\}$.
*C'est l'intersection de chaque élément de* $\Pi_X(r)$ *avec chaque élément de* $\Pi_Y(r)$.

**Propriété du produit de DV-Partitions :**

$$\Pi_X(r) \bullet_{dv} \Pi_Y(r) = \Pi_Y(r) \bullet_{dv} \Pi_X(r) = \Pi_{X \cup Y}(r).$$

**Exemple 5** *Le produit des deux DV-Partitions* $\Pi_{Seller}(r)$ *et* $\Pi_{Category}(r)$ *est le suivant :*
$\Pi_{Seller}(r) = \{< \{1,2,3\}, ???? >, < \{1,2\}, J??? >, < \{3\}, E??? >\}$
$\Pi_{Category}(r) = \{< \{1,2,3\}, ???? >, < \{1\}, ?C?? >, < \{2,3\}, ?S?? >\}$ *d'où*
$\Pi_{Seller}(r) \bullet_{dv} \Pi_{Category}(r) = \{< \{1,2,3\}, ??? >, < \{1,2\}, J??? >, < \{3\}, E??? >, < \{1\}, ?C?? >, < \{2,3\}, ?S?? >, < \{1\}, JC?? >, < \{2\}, JS?? >, < \{3\}, ES?? >\} = \Pi_{Seller,Category}(r)$

Nous présentons dans la section suivante un algorithme utilisant notre caractérisation pour effectuer un pré-calcul de cubes de données.

## 3.2 RSCube Algorithme

L'algorithme, RSCube (Reusable Structure of Cube), que nous proposons s'appuie sur la caractérisation présentée dans le paragraphe 3.1. RSCube, algorithme récursif, est basé sur une stratégie "diviser pour régner". La philosophie générale de cet algorithme est de découper récursivement la relation en deux parties suivant l'ensemble des dimensions jusqu'à la réduction à une seule. L'algorithme calcule alors la DV-Partition correspondant à la dimension. Les appels récursifs se terminent et on commence la remontée dans l'arbre binaire d'appels ainsi créé. A chaque niveau, RSCube calcule le produit des deux DV-Partitions (cf. Définition 5) jusqu'à la racine de l'arbre des appels. La figure 2 illustre le parcours de l'exécution de RSCube pour une relation à 8 dimensions.

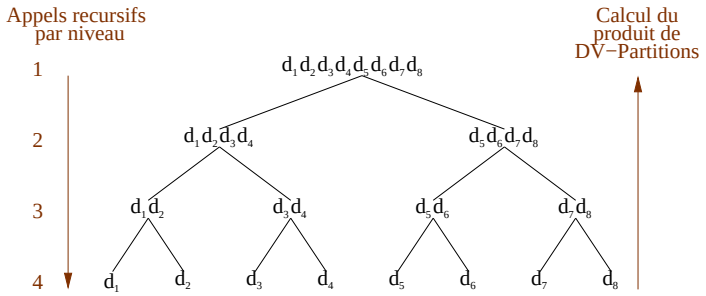


**FIG. 2** – *Arbre binaire de parcours de RSCube pour une relation à 8 dimensions {*$d_1$, $d_2$, ...,$d_8$*}*

La mise en œuvre de cette stratégie nous permet de découper les calculs de façon simple et de pouvoir les répartir sur plusieurs unités de calcul (multi-threads, multi-processeurs ou

plusieurs ordinateurs en réseau). Les calculs de produits de DV-Partitions étant réalisés en remontant, l'approche reste pour les calculs en Bottom-Up ce qui nous permet d'appliquer si besoin des méthodes d'élagages comme APriori.

Cet algorithme permet de pré-calculer le cube sur une ou plusieurs machines en réseau[1]. Dans le cas où une seule machine réalise la totalité des calculs, aucun transfert réseau n'a lieu et l'identifiant de la machine reste toujours le même.
Tous les calculs sont réalisés en tenant compte de la mémoire disponible sur chaque ordinateur suivant la configuration choisie par l'utilisateur. Pour cela, nous utilisons la fonction $ProductWithMemoryConstraint$ (ligne 8 de RSCube) qui calcule le produit des deux DV-Partitions en tenant compte de la mémoire disponible sur la machine qui exécute l'opération. Si la mémoire de la machine n'est pas suffisante, la DV-Partition résultant peut être partiellement calculée et donc certaines DV-Classes ne sont pas définies sur cette machine. Le reste du calcul devra être réalisé sur un autre ordinateur (cf. Figure 3). Dans le paragraphe 3.4, nous montrons comment nous réutilisons les fragments de DV-Partitions.

**Algorithm 1** RSCube($m$, $r$, $X$)

**Entrée :**
$m$ : L'identifiant de la machine de calcul
$r$ : Relation ou sous-relation
$X = \{d_1, d_2, ..., d_k\}$ : L'ensemble des dimensions de $r$
**Sortie :**
$\Pi_X(r)$ : La DV-Partition de $X$ pour la relation $r$

1: **if** $k = 1$ **then**
2: $\Pi_X(r) := \Pi_{d_1}(r)$
3: **else**
4: $X_1 := \{d_1, d_2, ..., d_{k/2}\}$
5: $X_2 := \{d_{k/2+1}, ..., d_k\}$
6: $\Pi_g(r) :=$ RSCube$(g, r[X_1], X_1)$
7: $\Pi_d(r) :=$ RSCube$(d, r[X_2], X_2)$
8: $\Pi_X(r) :=$ ProductWithMemoryConstraint$(\Pi_g(r), \Pi_d(r))$
9: **end if**
10: **return** $\Pi_X(r)$

**Exemple 6** *Désignons par A, B, C et D respectivement les dimensions Seller, Catégory, City et Custommer de la relation $CarSale$ (cf. Table 1). Notre approche calcule les DV-Partitions $\Pi_\emptyset(r)$, $\Pi_A(r)$, $\Pi_B(r)$, $\Pi_{AB}(r)$, $\Pi_C(r)$, $\Pi_D(r)$, $\Pi_{CD}(r)$, et $\Pi_{ABCD}(r)$. En effet nous divisons récursivement le problème en deux (en nombre de dimensions), en construisant les DV-Partitions de chaque sous problème, puis nous remontons à l'aide du produit des DV-Partitions pour construire le DVCube de la relation donnée. Une fois les DV-Partitions $\Pi_{AB}(r)$, $\Pi_{CD}(r)$ calculées, l'algorithme retourne le DVCube de ABCD ($\Pi_{ABCD}(r)$) en calculant $\Pi_{AB}(r) \bullet_{dv} \Pi_{CD}(r) = \Pi_{ABCD}(r)$.*

[1]Nous supposons que l'organisation réseau est connue par chaque ordinateur participant aux calculs.

*La représentation sous forme tabulaire du DVCube de la relation Car Sales est donnée dans le tableau 2.*

DVCube de *Car Sales*

| $t < X >$ | $S_t$ |
|---|---|
| ? ? ? ? | 1 2 3 |
| J ? ? ? | 1 2 |
| E ? ? ? | 3 |
| ?C ? ? | 1 |
| ?S ? ? | 2 3 |
| ? ?M ? | 1 2 3 |
| ? ? ?Y | 1 3 |
| ? ? ?A | 2 |
| JC ? ? | 1 |
| JS ? ? | 2 |
| ES ? ? | 3 |
| J ?M ? | 1 2 |
| E ?M ? | 3 |
| J ? ?Y | 1 |
| J ? ?A | 2 |
| E ? ?Y | 3 |
| ?CM ? | 1 |
| ?SM ? | 2 3 |
| ... | |

| $t < X >$ | $S_t$ |
|---|---|
| ?C ?Y | 1 |
| ?S ?Y | 3 |
| ?S ?A | 2 |
| ? ?MY | 1 3 |
| ? ?MA | 2 |
| JCM ? | 1 |
| JSM ? | 2 |
| ESM ? | 3 |
| JC ?Y | 1 |
| JS ?A | 2 |
| JS ?Y | 3 |
| J ?MY | 1 |
| J ?MA | 2 |
| E ?MY | 3 |
| ?CMY | 1 |
| ?SMA | 2 |
| ?SMY | 3 |
| JCMY | 1 |
| JSMA | 2 |
| ESMY | 3 |

TAB. 2 – *DVCube pour la relation initiale* Car sales

## 3.3 Stockage d'un DVCube

Le DVCube est la DV-Partition de toute la relation, il occupe donc $n2^{|Dim|}$ entiers pour représenter l'ensemble des identifiants des tuples des DV-Classes qu'il contient. De plus, il faut représenter les combinaisons de valeurs correspondantes. Si la relation initiale est accessible, on utilise directement l'indexation inhérente aux identifiants des tuples pour obtenir les valeurs dans la relation initiale ; si la relation n'est pas accessible, il faut converser le dictionnaire de la relation. Afin de simplifier l'étude, on suppose que la relation est accessible et donc un DVCube occupe $n2^{|Dim|}$ entiers.

Lorsque le DVCube (par exemple >1 To) ne peut pas être conservé sur une seule machine, nous pouvons le répartir sur plusieurs en réseau. Pour cela, nous répartissons les fragments du DVCube en fonction de leur taille et de celle des machines sur le réseau de façon à ce que la taille total des fragments de DVCube sur chaque machine soit proportionnelle à sa capacité de stockage. Bien sûr, d'autres paramètres peuvent être pris en compte pour améliorer la répartition de fragments de DVCube sur le réseau (vitesse du processeur, taille mémoire RAM...). La figure 3 illustre une fragmentation de DVCube et la répartition des fragments sur 3 machines en réseau. Cette technique de répartition des fragments de DVCube dans un réseau présente plusieurs avantages :

(i) Elle permet de calculer des cubes de données beaucoup plus volumineux et offre une alternative pour repousser les limites de l'espace de stockage.

(ii) Elle permet de calculer rapidement, en parallèle sur plusieurs machines, des cubes de données. Si besoin est, chaque machine calcule les valeurs de mesures du cube à partir de ses fragments de DVCube et renvoie le résultat à la machine source. Avec un réseau de $m$ machines identiques, les temps de calculs des cubes de données à partir d'un DVCube peuvent être divisés par $10m$ (cf. Section 4).

(iii) Par conséquence de (ii), même pour des cubes de données pouvant être stockés sur une machine, on peut envisager de répartir les fragments du DVCube sur un réseau afin d'améliorer les temps de calcul lors de la réutilisation du DVCube.

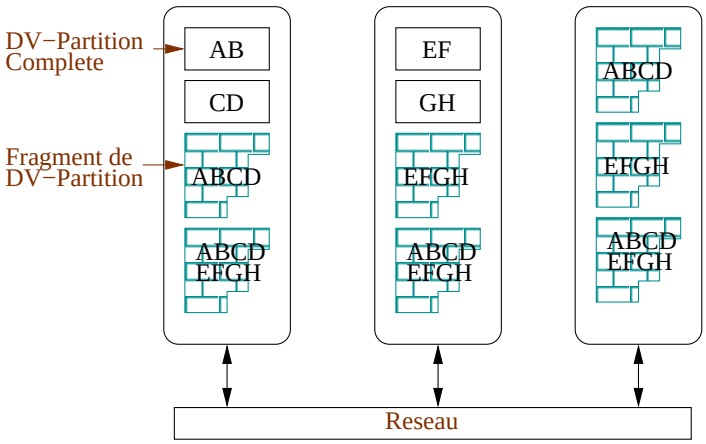


**FIG. 3** – *Répartition sur 3 machines des fragments d'un DVCube pour une relation à 8 dimensions*

## 3.4 Réutilisation du pré-calcul de cubes

La réutilisation du DVCube permet notamment de calculer plusieurs cubes de données simultanément c'est-à-dire pour plusieurs fonctions agrégatives. De plus, le précalcul facile l'interaction et la navigation dans le cube de données correspondant en permettant les calculs de fonctions agrégatives à la demande pour certains cuboïdes. Enfin, un nouveau domaine d'application des cubes de données concernent les données issues de systèmes automatisés ou plus simplement de capteurs. De telles données, comme les données météorologiques, ne varient que pour les valeurs mesures associées aux dimensions (par exemple des positions de capteurs de pluie, de vent...). Là encore, la réutilisation du DVCube permet un calcul efficace des valeurs agrégées correspondantes aux nouvelles mesures.

Sur la table 3, le DVCube est représenté entre les doubles barres verticales qui le séparent des valeurs agrégées associées aux cubes de données correspondant à la relation $CarSales$ (cf. Table 1). A partir du DVCube, tout ou partie des cubes de données est obtenu en calculant les fonctions agrégatives sur chaque DV-Class demandée. Pour chaque DV-Class, ce calcul est clairement linéaire au nombre d'élément de $S_t$ ($\mathcal{O}(|S_t|)$). Pour le cuboïde $AC$ dont $t < AC >= J?M?$ et $S_t = \{1, 2\}$ avec la fonction somme, nous obtenons la valeur agrégée 30.

| DVCube | | | Agrégations | | |
|---|---|---|---|---|---|
| | $t < X >$ | $S_t$ | $count$ | $somme$ | |
| $\emptyset$ | ? ? ? ? | 1 2 3 | 3 | 60 | |
| ... | ... | ... | ... | ... | *Moyenne* |
| D | ? ? ?Y | 1 3 | 2 | 40 | 20 |
| | ? ? ?A | 2 | 1 | 20 | 20 |
| ... | ... | ... | ... | ... | *Minimum* |
| AC | J ?M ? | 1 2 | 2 | 30 | 10 |
| | E ?M ? | 3 | 1 | 30 | 30 |
| ... | ... | ... | ... | ... | |
| BC | ?CM ? | 1 | 1 | 10 | |
| | ?SM ? | 2 3 | 2 | 50 | |
| ... | ... | ... | ... | ... | |

**TAB. 3** – *Cubes de Données complets pour les fonctions* count *et* somme *calculés à partir du DVCube de la relation* Car Sales *ainsi que pour les fonctions* Moyenne *et* Minimum *à la volée*

On procède de même pour tous les calculs en fonction des demandes de l'utilisateur (calcul à la volée). Pour les fonctions agrégatives algébriques, il est touefois possible d'améliorer la complexité du calcul en utilisant les résultants précedants. Les valeurs $Moyenne$ de 20 et 20 (respectivement) pour le cuboïde $D$ sont obtenues soit en recalculant la moyenne pour les tuples 1 et 3 (respectivement 2), soit en divisant simplement la somme (40, respectivement 20) par le nombre d'éléments (2, respectivement 1) ce qui réduit concidérablement la complexité ($\mathcal{O}(1)$). De même pour les valeurs minimales calculées suivant le cuboïdes $AC$.

# 4 Expérimentations

Les différentes expérimentations ont été réalisées sur un PC équipé d'un bi-processeur Xeon à 3 GHz avec 2 Go de RAM sous linux. La programmation a été faite en C++ (g++ (GCC) 4.0.2). Ces expérimentations montrent le bon comportement de l'algorithme RSCube ainsi que l'efficacité de la réutilisation du pré-calcul des cubes. Pour cela, nous avons utilisé deux types de jeux de données (synthétiques de grandes taille et réelles de petites tailles) :

- Données synthétiques afin de modifier à souhait les paramètres des relations dans le but d'observer le comportement de notre algorithme pour plusieurs situations. Nous avons généré des relations à 8 dimensions et 30% de corrélation puis nous avons fait varier le nombre de tuples de 200 000 à 800 000. Le générateur que nous avons utilisé, génère des données aléatoires suivant une distribution uniforme. Le taux de correlation correspond à la redondance de données.
- Données réelles pour tester l'efficacité de notre approche dans une situation réelle, mais aussi pour comparer les performances de notre approche avec celles de PCUBE (Casali et al. (2006)) dans le calcul de cubes de données (cf. Tableau 4).

Le tableau 4 montre que le calcul d'un DVCube est sensiblement plus rapide que le calcul d'un cube de données par les approches actuelles. Ces résultats ne sont pas surprenants

| *Jeux de données* | *#dimensions* | *# tuples* | *PCube* | *RSCube* | *RSCube$_{ht}$* | *Nb d'Agrégations* |
|---|---|---|---|---|---|---|
| NecropoleTombe | 7 | 1 846 | 15 ms | 12 ms | 7 ms | 113 721 |
| ObjetsDansTombe | 12 | 8 278 | 2587 ms | 1564 ms | 920 ms | 17 203 591 |

**TAB. 4** – *Comparaison des temps de calcul entre* PCUBE *et RSCube sur les jeux de données réelles*

puisqu'un DVCube est un pré-calcul de Cube de Données même si ces deux approches ont exactement le même nombre de lignes. De façon générale, RSCube obtient de meilleurs résultats que PCUBE. Bien entendu, ces deux approches ne sont pas conçues pour les mêmes objectifs.
De plus, nos expérimentations montrent l'efficacité de notre algorithme RSCube de calcul parallèle d'un DVCube : les résultats de la table 4 et de la figure 4 montrent qu'en utilisant deux threads sur une machine à deux processeurs, les temps de calculs sont pratiquement divisés par 2. Et enfin, nous montrons l'efficacité de la réutilisation d'un DVCube pour calculer un cube de données. La figure 5 montre que le temps de calcul d'un cube de données à partir d'un DVCube est presque 10 fois plus rapide que le temps de calcul du DVCube lui-même. Par conséquent le temps de calcul d'un cube de données à partir d'un DVCube est beaucoup plus rapide que le temps de calcul d'un cube de données par les approches actuelles.

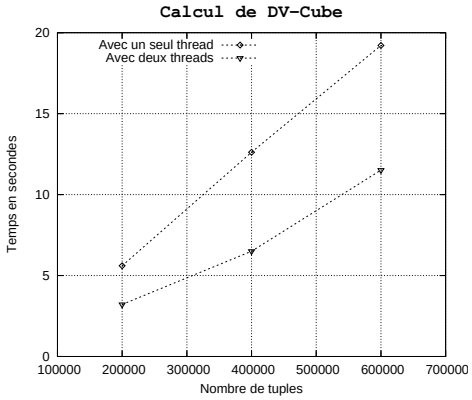


FIG. 4 – *Temps de calcul du DVCube*

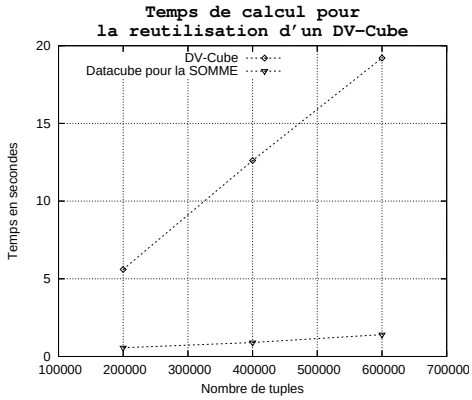


**FIG. 5** – *Comparaison des temps de la réutilisationdu pré-calcul et du calcul du cube de données*

# 5 Conclusion et perspectives

Nous avons proposé la notion de pré-calcul de cubes de données et une structure réutilisable associée appelée DVCube, sur laquelle s'appuie l'algorithme nommé RSCube, qui permet de calculer rapidement plusieurs cubes de données, en parallèle sur une ou plusieurs machines. Cette structure convient très bien pour résoudre la problématique liée à l'espace de stockage en permettant de calculer rapidement et à la volée toute valeur agrégée du cube pour toute fonc-

tion agrégative. Elle est particulièrement intéressante si l'analyse porte sur plusieurs critères (somme, moyenne, fréquence...) ou se fait graphiquement. Comme la visualisation devient très vite difficile voire impossible pour un cube en entier, une structure telle qu'un DVCube permet de traiter efficacement ce type de problème, puisqu'on peut à tout moment sélectionner et représenter rapidement toute portion du cube pour toute fonction agrégative.
Les expérimentations que nous avons menées montrent l'efficacité des calculs de notre approche, avec ou sans contrainte de mémoire face aux approches telles que PCUBE. La réutilisabilité de notre approche permet de ne pas calculer la totalité des agrégations du cube de données et de réaliser ces calculs à la demande de l'analyste. Malheureusement, nous n'avons pas pu nous comparer avec des approches utilisant la parallélisation faute de binaire fourni par les auteurs et par manque de temps pour les développer nous-mêmes.
Notre première extension concerne le calcul des cubes incomplets en utilisant un élagage. Cette technique courament utilisée en fouille de données et applicable pour le DV-Cube puisque les calculs sont effectués de bas en haut (Bottom-Up). Nous nous intéressons également à des travaux réduisant les besoins mémoire de notre approche en adaptant les techniques de résumé de cubes de données comme Quotient Cube ou Cube Lattices tout en conservant une réutilisabilité efficace. Une telle structure gardera la sémantique du cube sans perte d'information et réduira le besoin en mémoire et en stockage tout en rendant la manipulation du cube beaucoup plus aisée. Pour arriver à cette fin, nous orienterons aussi nos recherches vers le calcul de hiérarchies incluses dans un cube de données et de fait dans un DV-Cube afin d'en réduire la taille et les temps de construction et de réutilisation.

**Remerciements**

Ce travail a été réalisé dans la cadre de l'ACI JC 905 : "Cube de Données : Construction et Navigation Interactive."
Nous remercions les lecteurs anonymes pour leurs commentaires constructifs.

# Références

Barbará, D. et X. Wu (2000). Using loglinear models to compress datacube. In *Web-Age Information Management*, pp. 311–322.

Beyer, K. S. et R. Ramakrishnan (1999). Bottom-up computation of sparse and iceberg cubes. In *SIGMOD Conference*, pp. 359–370.

Casali, A., R. Cicchetti, et L. Lakhal (2003a). Cube lattices : A framework for multidimensional data mining. In *SDM*, pp. 304–308.

Casali, A., R. Cicchetti, et L. Lakhal (2003b). Extracting semantics from data cubes using cube transversals and closures. In *KDD*, pp. 69–78.

Casali, A., R. Cicchetti, L. Lakhal, et N. Novelli (2006). Lossless reduction of datacubes. In *17th DEXA*, Volume 4080 of *Lecture Notes in Computer Science*, pp. 409–419.

Chen, Y., F. K. H. A. Dehne, T. Eavis, et A. Rau-Chaplin (2004). Parallel rolap data cube construction on shared-nothing multiprocessors. *Distributed and Parallel Databases 15*(3), 219–236.

Cicchetti, R., N. Novelli, et L. Lakhal (2001). Apic : An efficient algorithm for computing iceberg datacubes. In *17ème conférence BDA*, pp. 229–242.

Colossi, N. G., W. Malloy, et B. Reinwald (2002). Relational extensions for olap. *IBM Systems Journal 41*(4), 714–731.

Cosmadakis, S., P. Kanellakis, et N. Spyratos (1986). Partition Semantics for Relations. *Journal of Computer and System Sciences 33*(2), 203–233.

Dehne, F. K. H. A., T. Eavis, S. E. Hambrusch, et A. Rau-Chaplin (2001b). Parallelizing the data cube. In *ICDT*, pp. 129–143.

Dehne, F. K. H. A., T. Eavis, et A. Rau-Chaplin (2001a). A cluster architecture for parallel data warehousing. In *CCGRID*, pp. 161–168.

Eisenberg, A. et J. Melton (2000). Sql standardization : The next steps. *SIGMOD Record 29*(1), 63–67.

Goil, S. et A. N. Choudhary (1998). High performance multidimensional analysis of large datasets. In *DOLAP, ACM First International Workshop on Data Warehousing and OLAP*, pp. 34–39.

Goil, S. et A. N. Choudhary (1999). A parallel scalable infrastructure for OLAP and data mining. In *International Database Engineering and Application Symposium*, pp. 178–186.

Gray, J., A. Bosworth, A. Layman, et H. Pirahesh (1996). Data cube : A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In *ICDE*, pp. 152–159.

Han, J., J. Pei, et Y. Yin (2000). Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, pp. 1–12. ACM.

Huhtala, Y., J. Karkkainen, P. Porkka, et H. Toivonen (1999). TANE : An Efficient Algorithm for Discovering Functional and Approximate Dependencies. *The Computer Journal 42*(2), 100–111.

Jermaine, C., A. Dobra, A. Pol, et S. Joshi (2005). Online estimation for subset-based sql queries. In *VLDB*, pp. 745–756.

Johnson, T. et D. Shasha (1997). Some approaches to index design for cube forest. *IEEE Data Eng. Bull. 20*(1), 27–35.

Karloff, H. J. et M. Mihail (1999). On the complexity of the view-selection problem. In *PODS*, pp. 167–173.

Lakshmanan, L. V. S., J. Pei, et J. Han (2002). Quotient cube : How to summarize the semantics of a data cube. In *VLDB*, pp. 778–789.

Lakshmanan, L. V. S., J. Pei, et Y. Zhao (2003a). Efficacious data cube exploration by semantic summarization and compression. In *VLDB*, pp. 1125–1128.

Lakshmanan, L. V. S., J. Pei, et Y. Zhao (2003b). Qc-trees : An efficient summary structure for semantic olap. In *SIGMOD Conference*, pp. 64–75.

Li, X., J. Han, et H. Gonzalez (2004). High-dimensional olap : A minimal cubing approach. In *VLDB*, pp. 528–539.

Lopes, S., J. Petit, et L. Lakhal (2000). Efficient Discovery of Functional Dependencies and Armstrong Relations. In *Proceedings of EDBT*, pp. 350–364.

Lu, H., X. Huang, et Z. Li (1997). Computing data cubes using massively parallel processors. In *Proceedings of the 7th Parallel Computing Workshop (PCW'97)*, Canberra, Austrilia.

Ng, W.-K. et C. V. Ravishankar (1997). Block-oriented compression techniques for large statistical databases. *IEEE KDE 9*(2), 314–328.

Novelli, N. et R. Cicchetti (2001a). Functional and embedded dependency inference : a data mining point of view. *Information Systems 26*(7), 477–506.

Novelli, N. et R. Cicchetti (2001b). FUN : An Efficient Algorithm for Mining Functional and Embedded Dependencies. In *Proceedings of ICDT'01*, Volume 1973 of *Lecture Notes in Computer Science*, London, UK, pp. 189–203.

Ross, K. A. et D. Srivastava (1997). Fast computation of sparse datacubes. In *VLDB*, pp. 116–125.

Saint-Paul, R., G. Raschia, et N. Mouaddib (2005). General purpose database summarization. In *VLDB*, pp. 733–744.

Shanmugasundaram, J., U. M. Fayyad, et P. S. Bradley (1999). Compressed data cubes for olap aggregate query approximation on continuous dimensions. In *KDD*, pp. 223–232.

Sismanis, Y. et N. Roussopoulos (2003). The dwarf data cube eliminates the high dimensionality curse. Technical report.

Spyratos, N. (1987). The Partition Model : a Deductive Database Model. *ACM Transactions on Database Systems 12*(1), 1–37.

Vitter, J. S., M. Wang, et B. R. Iyer (1998). Data cube approximation and histograms via wavelets. In *CIKM*, pp. 96–104.

Wang, W., H. Lu, J. Feng, et J. X. Yu (2002). Condensed cube : An efficient approach to reducing data cube size. In *ICDE*, pp. 155–165.

Zhao, Y., P. Deshpande, et J. F. Naughton (1997). An array-based algorithm for simultaneous multidimensional aggregates. In *SIGMOD Conference*, pp. 159–170.

# Summary

Datacubes are more and more used for the precalculation of OLAP queries in order to allow to analysts to find tendencies or anomalies in huge amounts of data. Its exponential complexity on almost every aspect has generated a flurry of research on a wide-variety of topics. In this paper, we introduce a new notion which is the precalculation of datacubes. To our knowledge, none of the previous approaches was interested in reusing a datacubes precalculation. However this reuse allows computing and manipulating efficiently datacubes in many cases like weather forecast applications, on fly computing queries or several datacubes in a network.