Fenêtres sur cubes

Yoann Pitarch, Anne Laurent, Marc Plantevit, Pascal Poncelet

FENÊTRES SUR CUBE

Yoann Pitarch ⋆ *Anne Laurent* ⋆ *Marc Plantevit* ⋆ *Pascal Poncelet* ⋆⋆

⋆ LIRMM, Université Montpellier 2
34090 Montpellier, France
(pitarch,laurent,plantevi)@lirmm.fr
⋆⋆ LGI2P, Ecole des Mines d'Alès
30035 Nîmes cedex 1, France
pascal.poncelet@ema.fr

RÉSUMÉ

Avec le développement des nouvelles technologies, de nombreuses applications (e.g. surveillance en temps réel, analyse du trafic, analyse de transactions financières, ...) doivent faire face à un flot éventuellement infini de données multidimensionnelles. Dans un contexte aussi dynamique, il n'est plus possible pour le décideur d'utiliser toutes les données à un faible niveau de granularité et il devient indispensable de proposer de nouvelles approches d'agrégation prenant en compte ces différentes contraintes. Dans cet article, nous adaptons les technologies OLAP à un contexte temps réel pour proposer une structure qui (1) permette une analyse multidimensionnelle et multi-niveaux efficace et (2) satisfasse une contrainte critique dans les flots de données : l'espace de stockage. Traditionnellement, l'historique des données de granularité faible n'est consulté que sur un passé proche et vouloir les stocker après ce délai devient superflu. Par conséquent, nous proposons de les agréger en fonction de l'évolution du flot au cours du temps passé en étendant le principe de fenêtres temporelles à toutes les dimensions hiérarchisées et introduisons les fonctions de précision pour déterminer à quel moment un niveau de granularité devient superflu. Ces fonctions sont ensuite combinées afin de proposer une structure compacte et rapidement maintenable. Les différentes expérimentations menées sur des jeux de données réels et synthétiques ont confirmé ces propriétés intéressantes.

***Mots-clés*—**OLAP, flot de données, résumé de données, fenêtres temporelles

1. INTRODUCTION

De nos jours, via les technologies OLAP, les cubes de données se sont imposés comme des composants essentiels dans de nombreux entrepôts de données participant à l'analyse des données et à l'aide à la décision [7, 1]. S'appuyant sur l'intégration et la consolidation de données multidimensionnelles, les techniques OLAP permettent une analyse en ligne puissante et rapide des mesures (e.g. des ventes) grâce à une matérialisation partielle ou totale sur plusieurs niveaux de granularité des dimensions d'analyse. Cependant aujourd'hui, ces technologies doivent faire face à un nouveau défi. En effet, le volume de données et surtout la vitesse à laquelle elles arrivent font que ces technologies ne sont plus adaptées. Le développement de nouvelles applications (e.g. gestion de trafic sur Internet, gestion de stock, échanges de transactions bancaires, ...) engendrent de gros volumes de données disponibles sous la forme de flot continu, potentiellement infini et de nature changeante. Par exemple, le nombre de transactions effectuées sur eBay est estimé aujourd'hui à plus d'un milliard par jour [5]. De la même manière, l'analyse de trafic Internet sur un réseau doit aujourd'hui considérer qu'il y a plus d'un million de paquets disponibles par heure et par routeur [12].

Ce changement de nature des données induit de nouvelles contraintes qu'il faut prendre en compte lors de la construction de cubes. En outre, quelle que soit la capacité de stockage et de traitement des ordinateurs, vouloir stocker et traiter toutes ces données est impossible. Il est donc indispensable que les approches proposées n'effectuent qu'une seule passe sur les données.

Le traitement de données disponibles sous la forme de flots est un problème de recherche récent [13] pour lequel il existe déjà de nombreuses contributions qui s'intéressent soit à la définition de nouveaux systèmes de gestion de flots (*Data Stream Management System*)[9, 2, 11] soit à l'extraction de connaissances (e.g. clustering [3, 10], recherche de motifs fréquents [16, 14], etc.) sur les flots.

Outre, les différentes contraintes énoncées, les flots de données multidimensionnelles doivent faire face au problème suivant : généralement, les données qui circulent sur le flot sont à un faible niveau de granularité et il est bien entendu impossible pour un décideur d'observer et d'analyser ces données sans que celles-ci ne soient agrégées.

Pour s'en convaincre, considérons le cas d'une étude de la consommation électrique d'une population. Il est irréaliste, dans un cadre d'aide à la décision (e.g. pour connaître les tendances générales de la population) de travailler sur les données brutes comme la consommation par minute et par compteur. Dans ce contexte, un niveau de précision intéressant pourrait être la consommation électrique par rue et par heure des habitants de la population étudiée. Ainsi, une question se pose : *comment agréger les données à un niveau de granularité suffisant pour pouvoir aider le décideur sachant que ces données sont disponibles sous la forme de flot et qu'on ne peut y accéder qu'une seule fois ?*

A notre connaissance, seule l'approche StreamCube [8] propose une solution pour construire et mettre à jour un cube alimenté par un flot et permettre ainsi une analyse multidimensionnelle et multi-niveaux.

Puisqu'il n'est pas possible de stocker avec un même niveau de précision tout l'historique d'un flot, le modèle de *tilted-time windows* (ou *fenêtres temporelles*) [6] est utilisé pour modéliser et compresser la dimension temporelle. Nous les présentons dans la section 2.3. Ce modèle s'inspire fortement du mécanisme d'oubli de la mémoire humaine et permet de stocker avec une précision maximale les données les plus récentes. En effet, plus les données vieillisssent, plus le niveau de précision diminue. Un tel modèle est donc tout à fait adapté à un contexte d'aide à la décision dans la mesure où le décideur est généralement intéressé par l'analyse des faits récents avec une grande précision mais veut malgré tout conserver une trace des données historiques pour par exemple évaluer les tendances. Pourtant, même si un tel modèle permet une compression importante de la dimension temporelle, il reste impossible de matérialiser les données sur toutes les dimensions et à tous les niveaux de précision, i.e. matérialiser l'intégralité du cube. Pour répondre à cette problématique, [8] propose une matérialisation partielle du cube en fixant deux cuboïdes particuliers appelés *critical layers* : le *minimal observation layer* (*m-layer*) et le *observation layer* (*o-layer*). Dans un contexte multidimensionnel hiérarchisé, un cuboïde représente un niveau de précision pour observer les données du cube. Par exemple, si un cube est défini sur deux dimensions (*Produit* et *Géographique*), un cuboïde possible serait (*CatProduit*, *Pays*). L'ensemble des cuboïdes forme le cube (appelé aussi *treillis des cuboïdes*). Le choix du *m-layer* se justifie par l'intuition que les données du flot arrivent à un niveau de précision beaucoup trop fin pour être exploitées en l'état par les utilisateurs et il est donc inutile de matérialiser ces données brutes. Il correspond donc au niveau de précision maximal matérialisé par l'utilisateur pour observer le flot. Bien entendu, le choix de la granularité entraîne l'impossibilité de consulter l'historique du flot à un niveau plus fin que celui défini par le *m-layer*. Le *o-layer*, par contre, est défini comme le niveau courant d'analyse pour l'utilisateur et représente le cuboïde le plus souvent consulté permettant au décideur de détecter les tendances globales et les exceptions. De manière à pouvoir répondre aux

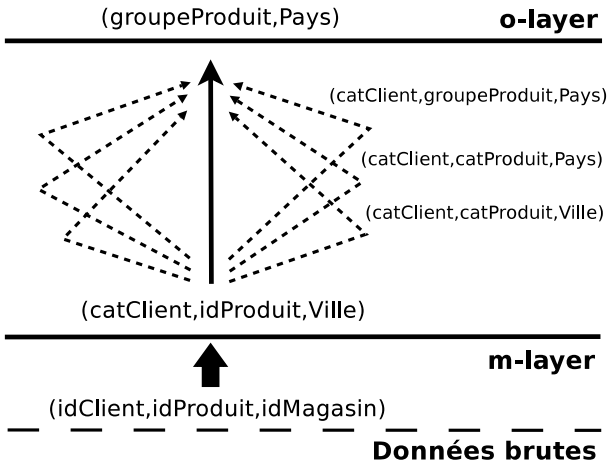


Fig. 1: Les différents niveaux de précision de [8]

requêtes OLAP avec suffisamment de précision et des temps de réponse acceptables, outre la détermination des deux *critical layers*, de nombreuses possibilités existent pour matérialiser ou pas les cuboïdes situés entre ces deux niveaux. [8] utilise pour cela le *popular path*. Il s'agit en fait d'un ensemble de cuboïdes formant un chemin dans le treillis reliant le *o-layer* au *m-layer*. Cet ensemble est déterminé à partir de requêtes habituellement formulées afin de minimiser les temps de réponses en réduisant les calculs nécessaires.

De manière à illustrer ces différents niveaux, considérons des données circulant sur le flot définies au niveau de précision *(idClient,idProduit,idMagasin)* (C.f. Figure 1).

Ce niveau étant jugé trop précis par l'utilisateur, il fixe alors *(catClient,idProduit,Ville)* comme étant le niveau de précision maximale pour consulter l'historique du flot (niveau *m-layer*). Ensuite, comme la plupart des interrogations concernent le cuboïde *(groupeProduit,Pays)*, celui-ci est choisi comme *o-layer*. Enfin les cuboïdes mentionnés entre les deux *critical layers* font partie du *popular path*.

Cette structure ouvre de nouvelles pistes de recherche. Malgré tout, cette proposition nous apparaît insuffisante dans un contexte aussi dynamique. Cette observation est motivée par les remarques suivantes :

– La propagation systématique des données du *m-layer* jusqu'au *o-layer* en suivant le *popular path* augmente le temps d'insertion d'une valeur dans la structure. Nous pensons que cette étape n'est pas obligatoire dans la mesure où l'opération de généralisation dans les cubes de données permet d'obtenir ces informations à partir du *m-layer* uniquement. En effet, nous considérons qu'il est acceptable pour un décideur de patienter quelques instants (le temps que la généralisation s'effectue) lorsqu'il interroge la structure. A l'inverse, il n'est pas envisageable que le temps de traitement d'une donnée (ou d'un ensemble de données) soit plus lent que le débit du flot. Cette situation entraînerait une perte de données et doit être absolument évitée.

– Malgré une matérialisation partielle du cube, le coût de stockage peut encore être réduit significativement sans perte majeure d'information. En effet, [8] stocke, pour chaque cuboïde matérialisé, l'intégralité de la *tilted-time window*. Ce choix n'est pas justifié car les décideurs ne consultent pas, pour chaque niveau, l'intégralité de l'historique.

Les principales contributions que nous proposons dans cet article sont les suivantes :

1. Généralement, l'historique des données de faible granularité n'est consulté que sur un passé récent. A partir de ce constat, nous proposons une technique inspirée des *tilted-time windows* pour ne matérialiser, sur chaque niveau de précision, que les périodes pendant lesquelles il est consulté. Cette approche permet de réduire significativement la quantité de données stockées pour chaque dimension.

2. Nous étendons l'idée précèdente au contexte multidimensionnel et proposons une structure compacte permettant une analyse multidimensionnelle et multi-niveaux efficace.

3. Un protocole de tests solide conduit sur des jeux de données synthétiques et réels souligne l'interêt de notre approche. En effet, même dans des conditions extrêmes, les résultats obtenus témoignent que notre structure est robuste et se maintient rapidement.

L'article est organisé de la manière suivante. La section 2 permet de définir les concepts de base et introduit la problématique. Notre proposition est décrite dans la section 3. Nous présentons et discutons les expérimentations et les résultats dans la section 4. Enfin, nous dressons quelques perspectives et concluons dans la dernière section.

2. DÉFINITIONS PRÉLIMINAIRES

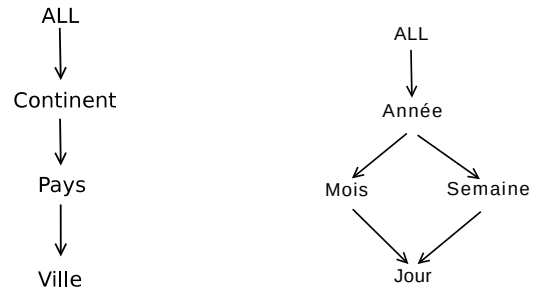
Dans cette section, nous proposons tout d'abord un rappel sur les cubes de données. Nous considérons ensuite les flots de données et définissons la notion de batchs. Avant de présenter la problématique, nous introduisons le modèle des *tilted-time windows*.

2.1. Cubes de données

Un cube de données est une application de n dimensions d'analyse $(D_1, ..., D_n)$ sur un ensemble de mesures $\mathcal{M}$ $(M_1, ..., M_k)$.

$$dom(D_1) \times .. \times dom(D_n) \rightarrow dom(M_1) \times .. dom(M_k)$$

Chaque dimension D_i est définie sur un ensemble (fini ou non) de valeurs noté $dom(D_i)$ et peut être observée à différents niveaux de granularité tels que *Ville*, *Département*, *Région* pour une dimension géographique. Ces niveaux constituent une hiérarchie de précision propre à chaque dimension D_i. Pour chacune d'elles, il existe une valeur joker (notée ALL_i ou $*$) qui peut être interprétée comme *toutes les valeurs de* D_i. Nous notons D_i^p le niveau p dans la hiérarchie de D_i (avec $0 \leq p \leq max_i$ et $D_i^0 = ALL_i$ le niveau le plus élevé) et avons $D_i^p > D_i^{p'}$ si D_i^p représente un niveau plus fin que $D_i^{p'}$. Nous notons $x \in dom(D_i^p)$ le fait qu'un élément est défini sur un niveau de hiérarchie p. A l'inverse, $Level(x)$ est utilisé pour spécifier dans quel niveau de hiérarchie x se trouve. Une hiérarchie pour laquelle un niveau se spécialise en un unique niveau est appelée hiérarchie *simple*. Autrement, elle est dite *complexe* (*i.e.* sa représentation forme un DAG (*Directed Acyclic Graph*)).



(a): Hiérarchie simple

(b): Hiérarchie complexe

Fig. 2: Exemples de hiérarchies

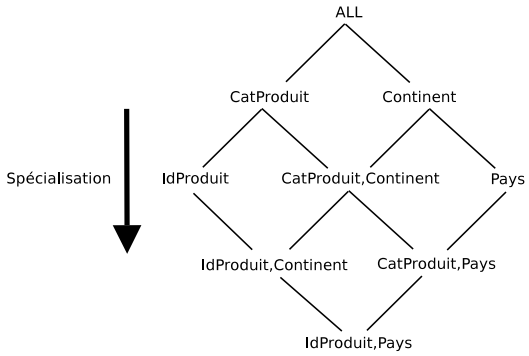


Fig. 3: Exemple de treillis

Exemple 1 *La figure 2 présente deux exemples de hiérarchies. La première 2(a) illustre une hiérarchie géographique simple où chaque niveau se spécialise en un unique niveau. La figure 2(b) est une hiérarchie complexe.*

Un cuboïde est un sous-ensemble de l'application définissant le cube. Formellement, un cuboïde M est défini par :

$$D_1^{l_1} \times D_2^{l_2} \times .. \times D_n^{l_2} \rightarrow M$$

avec $0 \leq l_i \leq max_i$ pour $1 \leq i \leq n$. L'ensemble des cuboïdes constitue un treillis et forme le cube de données.

Exemple 2 *Soient les dimensions Geo(Continent, Pays) et Produit(CatProduit, IdProduit). La figure 3 représente le treillis des cuboïdes correspondant.*

Soit $x \in dom(D_i)$ (avec D_i une dimension hiérarchisée). Nous notons $x^{\uparrow}$ l'ensemble des généralisations possibles de x selon la hiérarchie spécifiée. Notons que tout élément est inclus dans sa généralisation (*i.e.* nous avons $ALL_i^{\uparrow} = \{ALL_i\}$).

Exemple 3 *Si l'on reprend la dimension géographique de l'exemple précédent, nous avons :* $France^{\uparrow} = \{France, Europe, ALL\}$.

Pour une dimension D_i et un niveau de granularité p donné, nous définissons la fonction $p^{\sharp} : dom(D_i) \rightarrow dom(D_i)$ de la manière suivante :

$$p^{\sharp}(x) = \begin{cases} \emptyset & \text{si } D_i^p > Level(x) \\ x^{\uparrow} \cap dom(D_i^p) & \text{sinon} \end{cases}$$

Exemple 4 *Sur la même dimension géographique, nous avons :*

- $Continent^{\sharp}(France) = Europe$
- $Pays^{\sharp}(Asie) = \emptyset$.

2.2. Flots de données

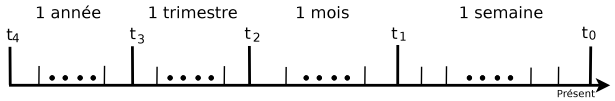
Un flot de données $S = B_0, B_1, ..., B_n$ est une séquence infinie de batchs (ensemble de tuples arrivant dans un même laps de temps) où chaque batch est estampillé avec une date t (B_t). B_n désigne le batch le plus récent.
Un batch $B_i = \{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3, ..., \mathcal{T}_k\}$ est défini comme un ensemble de n-uplets arrivant dans le flot pendant les $i^{ièmes}$ unités de temps. Dans un flot, la taille de chaque batch (i.e. le nombre de *n-uplets* les composant) peut être variable en fonction du mode de découpage choisi. Soit le découpage est effectué à intervalles de temps fixes auquel cas les batchs auront une taille différente si le débit du flot est variable, soit le flot est découpé selon un nombre de *n-uplets* fixe même si le débit du flot varie. Nous supposons que tous les batchs sont définis sur le même ensemble de dimensions D.

2.3. Tilted-time windows [6]

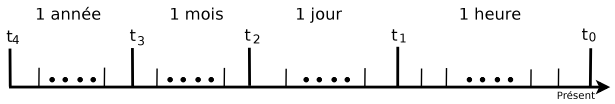
2.3.1. Présentation

Le caractère dynamique d'un flot de données entraîne l'impossibilité de matérialiser intégralement son historique. [6] propose donc le modèle des *tilted-time windows* pour résoudre cette difficulté en compressant les données sur la dimension temporelle. Dans un contexte toujours plus concurrentiel, la réactivité des décideurs doit être la plus grande possible. C'est pourquoi l'historique récent du flot doit pouvoir être étudié avec une précision temporelle maximale. A l'inverse, plus les données vieillissent, plus il devient acceptable de ne les conserver qu'à un faible degré de précision temporelle. Par conséquent, l'historique récent du flot est conservé avec un fin niveau de granularité. Ensuite, cette précision se dégrade de plus en plus en fonction de l'âge des données. Comme cela est illustré sur l'exemple suivant, la vitesse à laquelle les données se détériorent est variable selon domaine d'application. Nous notons $T = \langle t_0, t_1, ..., t_k \rangle$ une tilted-time de taille k avec t_0 pour désigner le présent. Chaque intervalle $[t_i, t_{i+1}]$ (avec $0 \leq i < k$) représente un niveau de précision et est appelé fenêtre.

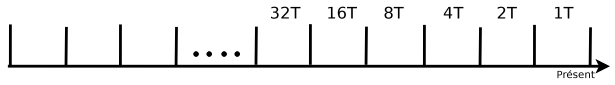
Exemple 5 *La figure 4 expose trois exemples de* tilted-time. *En effet, [6] propose deux modèles différents. Le premier (4(a) et 4(b)) est celui des fenêtres temporelles naturelles. Les niveaux de granularité sont fixés en fonction de l'application et suivent un découpage naturel du temps. Par exemple, si l'on considère les ventes d'un magasin, la figure 4(a) stockera pendant une semaine les ventes à la journée. Ensuite, pendant un mois, les ventes seront stockées à la semaine et ainsi de suite. Ce modèle permet une réduction non négligeable du nombre de valeurs stockées. Si l'on matérialisait toutes les ventes à la journée pendant un an, il faudrait stocker 365 valeurs. Le modèle des tilted-time illustré dans la figure 4(a) stockera seulement 18 valeurs (7+4+3+4). Le second modèle de tilted-time (4(c)) effectue un découpage logarithmique. Dans le cas d'un log en base 2 (comme sur la figure), pour représenter 365 valeurs il faudrait seulement* $log_2(365)$



(a): *Tilted-time* naturelle 1



(b): *Tilted-time* naturelle 2



(c): *Tilted-time* logarithmique

Fig. 4: Exemples de *tilted-time*

valeurs.

2.3.2. Insertion d'une valeur

Le débit d'un flot pouvant être très rapide, il est indispensable que le mécanisme d'insertion d'une nouvelle valeur du flot dans une tilted-time soit le plus simple et rapide possible. Pour plus de clarté, nous l'illustrons avec la figure 5. La situation initiale présente la *tilted-time* aprés l'insertion de deux mesures. Considérons maintenant la prochaine mesure arrivant du flot. 3 est inséré dans la première case de la *tilted-time*. Cette case étant occupée par 2, 2 est alors décalé vers la case de gauche. Là aussi cette case est occupée (par 1). 1 est alors décalé vers la gauche. Puisque, cette fois -ci, la case était libre, l'algorithme s'arrête et nous aboutissons à la situation T=1 de l'exemple. L'arrivée d'une nouvelle valeur (4) provoque une situation différente. En effet, la première fenêtre étant pleine, l'insertion de 4 entraîne dans un premier temps l'agrégation de la première fenêtre. La fonction d'agrégation choisie dans l'exemple est la somme. Ainsi, les opérations effectuées sont (1) insertion de 4 dans la première case de la première fenêtre temporelle et (2) insertion de la valeur agrégée dans la première case de la seconde fenêtre. Comme cette insertion se fait sur une case libre, l'algorithme s'arrête et nous aboutissons à la situation illustrée à T=2. Dans le cas contraire, le même mécanisme de décalage vers la gauche aurait été mis en place tant que l'on ne tombe pas sur une case vide.

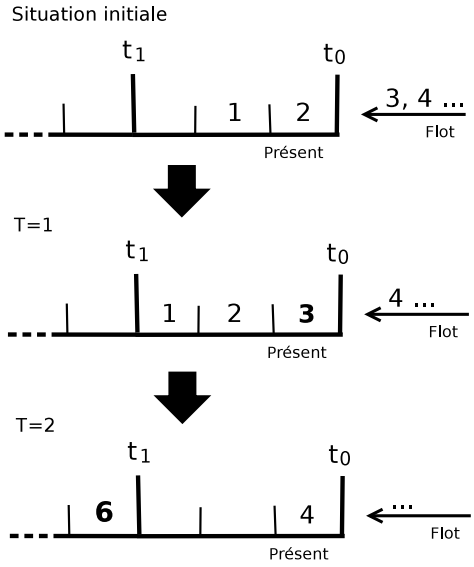


Fig. 5: Mécanisme d'insertion d'une valeur dans une tilted-time

2.4. Problématique

Les sections 2.1 et 2.2 témoignent de la nécessité de mettre en place une structure satisfaisant des contraintes multiples. Il est en effet impératif de conserver un historique de l'ensemble des données circulant dans le flot tout en garantissant que l'espace mémoire occupé soit le plus compact possible. Outre cette contrainte d'espace non-négligeable, deux contraintes temporelles doivent être prises en compte : (1)le temps d'insertion d'un batch dans la structure ne doit pas dépasser le délai d'arrivée entre deux batchs et (2) la structure doit permettre une interrogation efficace pour les requêtes les plus couramment formulées.

2.5. Exemple

Considérons le flot des ventes à la journée d'une chaîne de magasins spécialisée dans les nouvelles technologies du divertissement. Le tableau 1 présente un extrait d'un tel flot. Chaque n-uplet y circulant est observable au niveau de précision (*Produit*, *Ma-*

Date	Produit	Magasin	Vente
1	Wii	Montpellier	14
1	iPhone	Pekin	22
2	Mac Air	Tokyo	3
3	PS3	Paris	20
3	F490	Madrid	8
4	iPhone	Tokyo	200
5	D630	Montpellier	8
6	Wii	Pekin	40
6	Mac Air	Madrid	32
7	D630	Paris	10
8	Wii	Canton	5
9	Mac Air	Montpellier	35
10	PS3	Tokyo	100

Table 1: Extrait du flot multidimensionnel

gasin). La figure 6 présente les différents niveaux de précision possibles pour observer ces ventes et aider les décideurs à analyser ce flot. Il faut donc conserver l'historique de ces ventes sur plusieurs niveaux de précision mais la matérialisation de tous les cuboïdes serait trop coûteuse. Généralement les utilisateurs demandent que l'historique récent soit observable avec une précision maximale et accepte que cette précision se dégrade avec le temps. Nous utilisons donc la *tilted-time window* illustrée dans la figure 7 et étudions les fréquences d'interrogation de chaque niveau de précision en fonction de l'âge de l'historique (tableau 2). Nous observons que les précisions élevées sont majoritairement consultées sur l'historique récent du flot. Par exemple, le niveau *IdProduit* n'est consulté que sur la dernière semaine écoulée. Nous proposons donc une approche prenant en compte cette remarque en ne matérialisant, pour chaque niveau de précision, que les périodes pendant lesquelles il est consulté. Ainsi, notre méthode ne matérialisera que $[t_0; t_1]$ sur le niveau *Ville* garantissant une réduction du coût de matérialisation.

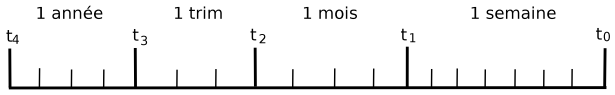


Fig. 7: *Tilted-time window* utilisée dans l'exemple

Niveau	Intervalle	Fréquence
DIMENSION GEO		
Ville	$[t_0, t_1]$	98%
	$[t_1, t_2]$	2%
Pays	$[t_0, t_1]$	25%
	$[t_1, t_2]$	74%
	$[t_2, t_3]$	1%
Continent	$[t_3, t_4]$	100%
DIMENSION PRODUIT		
IdProduit	$[t_0, t_1]$	100%
CatProduit	$[t_1, t_2]$	70%
	$[t_2, t_3]$	20%
	$[t_3, t_4]$	10%

Table 2: Fréquence d'interrogation des niveaux de précision

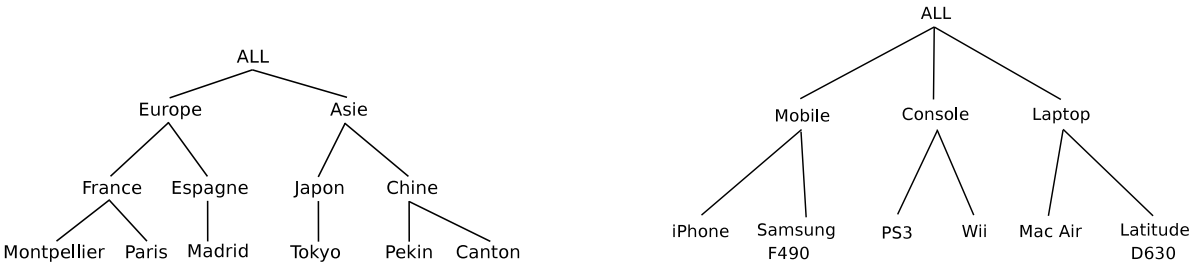
3. PROPOSITION

Dans cette section, nous présentons d'abord le concept central de notre proposition : les *fonctions de précision*. Nous étendons ensuite ces fonctions à l'ensemble des dimensions et présentons les algorithmes utilisés dans notre proposition.

3.1. Fonctions de précision

3.1.1. Idée générale

Dans le contexte des flots de données, la quantité de mémoire occupée par la structure proposée est une contrainte forte. Pour l'intégrer, [8] compresse l'historique du flot sur la dimension temporelle grâce aux *tilted-time windows*. Pour les raisons déjà évoquées dans la section 1, cette approche nous semble insuffisante. Nous proposons donc d'ét-



(a): Localisation (b): Produit

Fig. 6: Hiérarchies utilisées dans l'exemple

endre cette compression à toutes les dimensions hiérarchisées.

Pour nous convaincre de l'inutilité de stocker l'intégralité de l'historique pour chaque niveau de précision, observons le tableau 2. Nous constatons par exemple que le niveau *IdProduit* n'est consulté que sur le premier intervalle de la *tilted-time window*. Il est donc naturel de se demander pourquoi le reste de l'historique devrait-il être conservé à ce niveau de précision.
Nous proposons alors une structure réduisant l'espace de stockage en nous appuyant sur l'idée que les intervalles de temps stockés pour chaque niveau de la hiérarchie d'une dimension doivent tenir compte des habitudes des utilisateurs. En effet, un cube stockant les ventes par *IdProduit* pour le mois courant et généralisant ensuite sur le niveau supérieur (*CatProduit*) réduirait l'espace de stockage tout en garantissant des réponses précises aux requêtes courantes. Cet exemple illustre à la fois les avantages en terme d'espace qu'un tel système engendrerait mais aussi l'importance de définir correctement le moment où un niveau de précision ne sera plus stocké. Ces choix doivent nécessairement être faits par les utilisateurs du système. Sans cela, les conséquences pourraient être importantes en provoquant l'impossibilité de répondre à des requêtes souvent formulées. Les méthodes proposées pour déterminer ces choix sont exposés plus loin. Nous formalisons d'abord cette proposition en définissant les *fonctions de précision*. Nous illustrons ensuite notre proposition au travers d'un exemple. Enfin, nous présentons les méthodes mises en jeu pour respecter le compromis entre la minimisation de l'espace de stockage et la satisfaisabilité des requêtes usuelles puis faisons quelques remarques sur l'espace de stockage nécessaire.

3.1.2. Formalisation

Notre objectif est donc de définir une valeur fin qui, pour chaque niveau p d'une dimension D_i (D_i^p), indique le délai à partir duquel D_i^p n'est plus (ou presque plus) interrogé. Nous notons fin_i^p cette valeur pour le niveau D_i^p avec $fin^p \in t_1, .., t_k$ et $fin_i^1 = t_k$. En d'autres termes, fin_i^p (fin^p quand il n'y a pas d'ambiguïté sur les dimensions) doit correspondre à une extrémité d'une des fenêtres de la *tilted-time window* et nous fixons, pour chaque dimension D_i, fin_i^1 (*i.e.* le niveau le moins précis excepté ALL_i) égal à t_k (*i.e.* la limite de la tilted-time). Pour chaque dimension D_i, ALL_i n'est pas défini. Nous proposons dans un premier temps, de ne plus matérialiser pour chaque niveau fin_i^p, la partie de la tilted-time correspondante à $[fin_p; t_k]$ afin d'être fidèle à l'intuition décrite plus haut.
Cependant, la matérialisation de $[t_0; fin_p]$ pour chaque niveau conduirait à une forme de redondance. En effet, imaginons par exemple que $fin^{idProduit} = t_1$ et $fin^{catProduit} = t_4$. L'opérateur de généralisation OLAP permet d'obtenir par le calcul les données au niveau $catProduit$ sur l'intervalle $[t_0; t_1]$ sans avoir à les stocker.
Nous introduisons alors $debut^p = fin^{p+1}$ avec $1 \leq p < max_i$ et $debut^{max_i} = t_0$ ($debut^p$ quand il n'y a pas d'ambiguïté sur la dimension) pour limiter cette

redondance. Ces deux bornes permettent maintenant de définir une fonction bijective *Precision* qui, pour chaque niveau D_i^p, renvoie $[debut^p; fin^p]$ associé. $Prec(D_i^p)$ représente l'intervalle minimal en terme de stockage permettant de répondre aux requêtes des utilisateurs. Nous remarquons que $\{Prec(D_i^p),\ 0 \leq p \leq max_i\}$ est une partition de T pour une dimension donnée D_i.

Exemple 6 *La figure 8 illustre les fonctions de précision utilisées dans l'exemple. Par exemple,* $Prec(Pays) = [t_1; t_3]$ *signifie que (1) si l'on interroge ce niveau sur le dernier mois ou dernier trimestre, la réponse est instantanée car le résultat est déjà calculé et stocké ; (2) il ne sera pas possible de connaître le niveau des ventes par ville sur cette période. En effet, connaissant les ventes en France du dernier mois, il n'est pas possible d'en déduire les ventes sur* Montpellier *et* Paris*. Connaissant la distribution des ventes (e.g. que l'on vend deux fois plus de consoles à Paris etc..), il serait possible d'approcher les résultats mais sans garantir leur exactitude. (3) A l'inverse, il est tout à fait possible de connaître la quantité exacte des ventes sur cette période au niveau des continents. Ce résultat s'obtient sans difficulté grâce à l'opérateur de généralisation défini dans les cubes de données.*

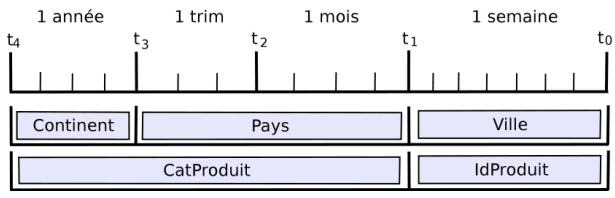


Fig. 8: Représentation graphique des précisions

Nous définissons la fonction inverse de $Prec$ qui, pour un niveau $[t_j; t_{j+1}]$ de la *tilted-time window* et une dimension D_i renvoie le niveau associé dans la hiérarchie et la notons :

$$Prec_i^{-1}([t_j; t_{j+1}]) = X$$

avec $Prec(X) = [t_k; t_l]$ et $k \leq j < l$.

Si l'on reprend les données de l'exemple précédent, nous avons $Prec_{Geo}^{-1}([t_0; t_1]) = Ville$.

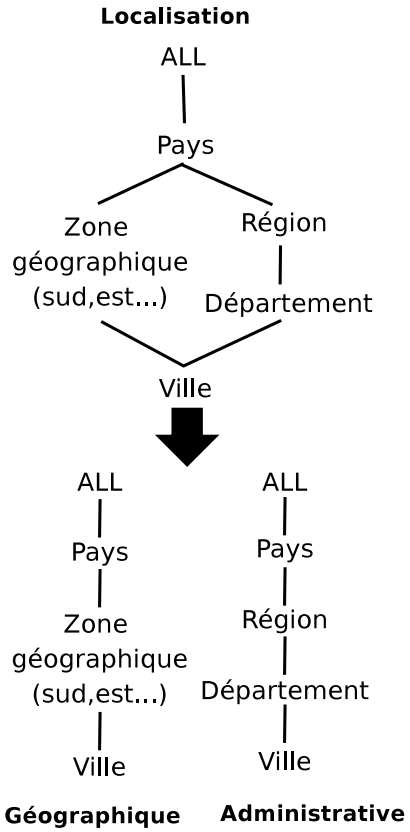


Fig. 9: Découpage d'une hiérarchie complexe

Remarque 1 *Jusqu'à présent, nous n'avons utilisé que des hiérarchies simples pour illustrer notre propos. Si des hiérarchies complexes sont définies, nous les découpons pour faire autant de hiérarchies simples que nécessaire comme l'illustre la figure 9.*

Remarque 2 *Dans le cas où un même niveau de précision intègre plusieurs attributs (par exemple code de ville et nom de ville), une solution consiste à fusionner les attributs sémantiquement équivalents et considérer ainsi un unique attribut.*

3.1.3. Mise à jour

Le modèle proposé s'inspirant fortement des *tilted-time windows*, sa mise à jour en reprend les principes. La seule différence est que, pour un niveau D_i^p, lorsque l'intervalle de la *tilted-time* défini par $Prec(D_i^p)$ est plein (*i.e.* il faut passer à un niveau de granularité plus faible), une double agrégation

s'opère. Classiquement, les mesures de la première fenêtre sont agrégées en une valeur v selon la fonction définie. Ensuite, toutes les valeurs v issues de cellules c de même niveau et dont les $(p-1)^{\uparrow}$ sont identiques (*i.e.* les cellules dont la généralisation sur le prochain niveau fournit la même valeur) sont agrégées. Le résultat de cette agrégation est alors stocké dans la première case de l'intervalle de la *tilted-time* de $(p-1)^{\uparrow}(c)$.

Exemple 7 *Supposons que nous ne nous intéressions qu'à la dimension geographique de l'exemple. La séquence à insérer dans la* tilted-time *de la cellule* Montpellier *est* $< 14, 0, 0, 0, 8, 0, 0, 0, 35, 0 >$. *Cette séquence peut se traduire ainsi : il y a eu 14 ventes dans le premier batch (jour) à Montpellier, aucune le second etc...La fonction d'agrégation utilisée ici est la* somme. *Selon le mécanisme classique des* tilted-time windows *et en supposant toutes les fenêtres initialement vides, les données sont insérées dans la première fenêtre de la tilted-time jusqu'à ce qu'elle soit pleine. Ainsi, après le batch 7, nous nous trouvons dans la situation illustrée dans la partie supèrieure de la figure 10. Le traitement du batch suivant implique le passage d'une fenêtre à une autre. Dans un premier temps, la première fenêtre de chaque cellule est agrégée en 22 pour Montpellier et 30 pour Paris. La seconde étape consiste à agréger entre-elles les valeurs dont la généralisation sur le niveau* Pays *conduit au même résultat. Comme* $Ville^{\uparrow}(Montpellier) = Ville^{\uparrow}(Paris) = France$, *22 est ajouté à 30. Enfin, le résultat est placé dans la première case de la* tilted-time *associée à* $France$ *et les valeurs du batch 8 (0 et 0) sont insérées dans les* tilted-time windows *correspondantes. Le résultat produit est illustré dans la partie infèrieure de la figure 10.*

Cet exemple illustre la nécessité de bien déterminer les fonctions de précision pour chaque dimension. Nous consacrons la prochaine sous-section à la description des méthodes adoptées pour leur définition.

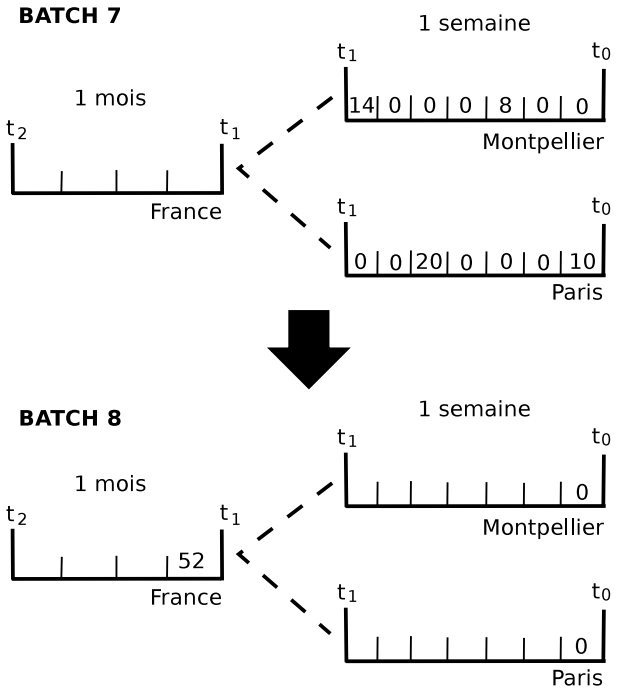


Fig. 10: Illustration d'un changement de fenêtre

3.1.4. Caractérisation des fonctions

Une bonne définition des fonctions de précision est cruciale pour le modèle afin de couvrir le plus de requêtes possibles. Ces fonctions doivent donc obligatoirement être déterminées par rapport aux préférences de l'utilisateur selon le domaine d'application et en tenant compte de ses requêtes habituelles. Pour ce faire, deux stratégies peuvent être adoptées :

1. Une stratégie naïve consisterait à laisser l'utilisateur déterminer manuellement ses préférences (e.g. à travers un formulaire générant un fichier XML). Cette solution s'avère cependant peu réaliste puisqu'il est rare qu'il n'y ait qu'un seul utilisateur et que par conséquent les préférences de l'utilisateur ayant renseigné les fonction de précision peuvent être inadaptées pour les autres utilisateurs.
2. Une solution avancée consisterait à définir automatiquement les fonctions de précision en utilisant des logs de requêtes et les statistiques sur les niveaux de précision interrogés en fonc-

tion du temps. Nous déterminons alors les t_{fin} de sorte à pouvoir répondre à un pourcentage fixé de requêtes du log. Notons que le cas où $fin_i^{max_i} = t_k$ (tous les intervalles matérialisés sur le niveau le plus précis) garantit un taux de réponse maximal mais engendrerait trop de généralisations pour satisfaire la plupart des requêtes. Pour éviter cette situation, nous nuançons la condition du taux de satisfaisabilité par *l'effort de généralisation*. Concrètement, si deux niveaux d'une dimension permettent de répondre au même nombre de requêtes, nous choisissons de matérialiser le niveau permettant de minimiser le nombre de calculs nécessaires pour satisfaire la requête.

3.1.5. Considérations sur l'espace de stockage

Nous quantifions ici le gain en terme d'espace de stokage engendré par notre proposition par rapport à [8]. Pour cela, nous utilisons la dimension *Geographique* (hiérarchie et domaine de définition) et la *tilted-time window* présentés dans l'exemple et étudions la quantité de données matérialisées en fonction de deux fonctions de précision définies ainsi :

1. Precision(Ville)=$[t_0; t_4]$, tous les intervalles sont stockés sur le niveau le plus fin. Nous notons cette configuration $prec_1$
2. Precision(Continent)=$[t_0; t_4]$, tous les intervalles sont stockés sur le niveau le moins précis. Nous notons cette configuration $prec_2$

Ces deux configurations vont permettre de fixer les bornes sur la quantité d'éléments matérialisés. Pour simplifier notre propos, l'hypothèse est faite que chaque intervalle de *tilted-time window* matérialisé vaut 1 unité.

Puisque $\mid Dom_{Ville} \mid = 6$, le coût de $prec_1$ serait 24 (6×4). Pour $prec_2$, $\mid Dom_{Continent} \mid = 2$. Par conséquent, son coût de matérialisation sera de 8.

Calculons maintenant le coût de l'approche StreamCube sur cet exemple. Nous considérons ici que $Ville$ appartient au *m-layer*, $Continent$ au *o-layer* et $Pays$ au *popular path*. Avec ces hypothèses et puisque [8] propose de matérialiser l'intégralité de la *tilted-time window* sur les niveaux énoncés, le coût de la matérialisation serait de 48. Nous remarquons ainsi que dans la pire des situations ($prec_1$), notre approche divise par 2 le nombre d'éléments matérialisés comparé à StreamCube. L'objectif ici n'est pas de montrer que nous avons tout intêret à matérialiser l'ensemble de l'historique sur le niveau le plus fin (ce choix doit être guidé par l'utilisateur) mais plutôt d'affirmer que, quelles que soient les fonctions de précision utilisées, notre approche réduit significativement l'espace nécessaire pour stocker l'historique du flot.

3.2. Extension à toutes les dimensions

Dans la section précédente, nous avons défini, pour chaque dimension, une fonction possédant la propriété de réduire l'espace de stockage nécessaire et satisfaisant les requêtes usuelles des utilisateurs. Nous nous intéressons maintenant à la méthode pour combiner ces fonctions. Pour ce faire, nous proposons une méthode simple utilisant l'opérateur d'intersection ensembliste et qui conserve les bonnes propriétés des fonctions de précision.
La figure 11 illustre la méthode proposée. Par exemple, nous constatons que *Ville* et *IdProduit* sont tous deux définis sur $[t_0; t_1]$. Par conséquent, matérialiser uniquement cet intervalle sur le cuboïde (*Ville*,*IdProduit*) ne contredirait pas les fonctions de précision des deux dimensions. Dans le cas où les fonctions de précision ne coincident pas, l'approche proposée respecte également les fonctions de précision prespectives. Par exemple, *Pays* et *CatProduit* partagent $[t_1; t_3]$ mais *CatProduit* partage également $[t_3; t_4]$ avec *Continent*. $[t_1; t_3]$ est donc matérialisé sur le cuboïde (*Pays*,*CatProduit*) et $[t_3; t_4]$ sur (*Continent*, *CatProduit*). Comme cela est décrit dans la figure 11, cette méthode conserve les fonctions de précision de chaque dimension.

Soit $C = D_1^{p_1} D_2^{p_2} .. D_k^{p_k}$ un cuboïde (avec $1 \leq k \leq n$ et $0 \leq p_i \leq P_{max_i}$ pour $1 \leq i \leq k$). La fonction $Materialize(C)$ est définie ainsi :

$$Materialize(C) = \begin{cases} \emptyset, & \text{si } k \neq n \\ \bigcap_{i=1}^{k} Prec(D_i^{p_i}), & \text{sinon} \end{cases}$$

Un cuboïde C sera matérialisé si et seulement si $Materialize(C) \neq \emptyset$ et chaque mesure de ce cuboïde sera alors une partie d'une tilted time window représentant $Materialize(C)$.

Exemple 8 *La figure 11 illustre le résultat de la proposition sur l'exemple. Seulement 3 cuboïdes du treillis sont matérialisés et ne stockent chacun qu'une partie de la* tilted-time.

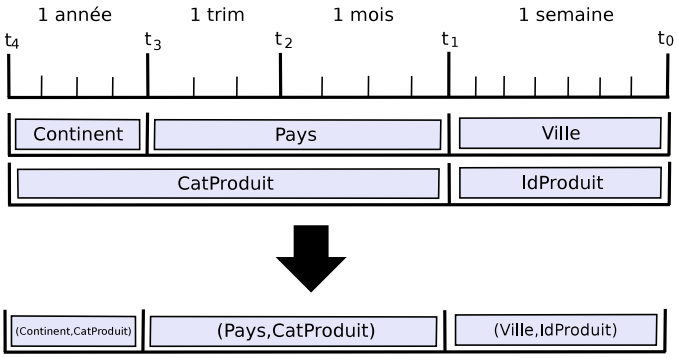


Fig. 11: Résultat de l'intersection des fonctions de précision

Propriété 1 *Par définition de l'intersection, cette proposition conserve les niveaux de précision définis par l'utilisateur pour chaque dimension. Il n'existe pas de cuboïde* $C = D_1^{p_1} D_2^{p_2} .. D_k^{p_k}$ *tel que* $Materialise(C) \nsubseteq Prec(D_i^{p_i})$ *pour tout* $D_i^{p_i} \in \{D_1^{p_1}, D_2^{p_2}, .., D_k^{p_k}\}$.

Nous réalisons ici une étude du nombre de cuboïdes matérialisés. Dans un contexte multidimensionnel hiérarchisé, la hauteur du treillis des cuboïdes, (notée H), est égale à la somme du nombre de niveau (excepté ALL) plus 1. Sur l'exemple, la dimension Geographique est observable sur 3 niveaux et Produit sur 2. Comme nous le voyons sur la figure 12, le treillis est bien de hauteur 6. Puisque par définition aucun intervalle n'est défini sur le niveau ALL, le nombre de cuboïdes matérialisés est borné par $\mathcal{H} - 1$ car le cuboïde le plus haut ne sera jamais matérialisé.

Nous voyons maintenant qu'il existe des situations où cette borne est encore plus petite. En effet, chaque intervalle de la *tilted-time window* n'est matérialisé que sur un cuboïde du treillis. Par conséquent, la pire des situations serait de matérialiser un intervalle par cuboïde. Par conséquent, une autre borne est la taille de la tilted-time windows (i.e. le nombre d'intervalle qui la compose noté $\mid T \mid$).

Par conséquent, le nombre de cuboïdes matérialisés par notre approche est $min(\mathcal{H} - 1, \mid T \mid)$. La figure 12 présente les cuboïdes matérialisés dans les conditions énoncées dans l'exemple.

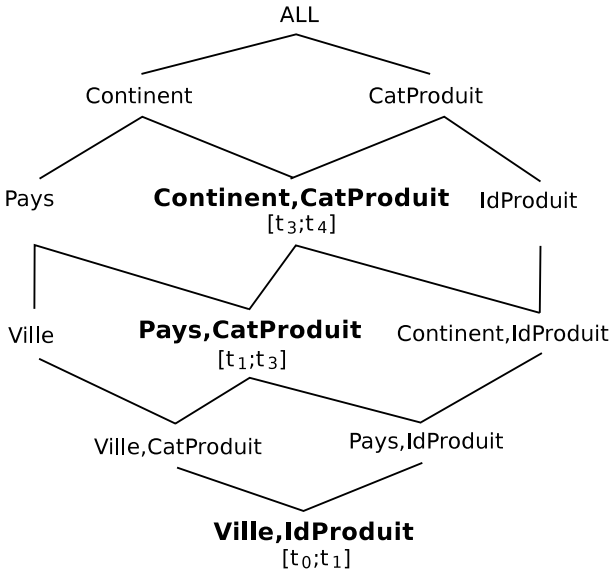


Fig. 12: Cuboïdes du treillis matérialisés

3.3. Algorithmes

Nous présentons ici les algorithmes développées pour cette proposition. Dans un premier temps, nous décrivons le fonctionnement général de la méthode et nous intéressons ensuite à une étape importante : l'insertion d'un n-uplet dans la structure.

L'algorithme général contient deux phases distinctes : l'initialisation et le traitement. La première consiste à (1) récupérer ou calculer les fonctions de précisions pour chaque dimension, (2) les combiner et (3) générer les batchs pour simuler un flot si nous nous plaçons dans le contexte de données synthétiques. Une fois la structure initialisée, il ne reste plus qu'à traiter les n-uplets arrivant dans le flot sous forme de batch. C'est l'étape d'insertion et la mise à jour de la structure. Dans un contexte aussi dynamique que celui de notre proposition, il est indispensable que cette étape soit la plus simple possible.

L'algorithme 1 décrit le fonctionnement général de la méthode et détaille les fonctions appelées pour la phase d'initialisation :

- *precision()* permet de calculer les fonctions de précisions décrite dans la section 3.1.4.
- *intersection()* calcule les cuboïdes à matérialiser et les intervalles de *tilted-time* qu'ils stockeront. Comme expliqué dans la partie 3.2, la fonction effectue l'intersection des fonctions de précision et stocke les résultats dans une structure globale.

Algorithme 1 : Général

Données : $nbBatchs$ le nombre de Batchs à générer, $sizeOfBatch$ le nombre de tuples dans un batch

```
1 début
    /* INITIALISATION                    */
2   precision() ;
3   intersection() ;
    /* TRAITEMENT                        */
4   pour i ∈ {0, 1, ..., nbBatchs − 1} faire
5     pour chaque cell ∈ BATCHi faire
6       insereValeur(cell,mes,1,i);
7     update(i);
8 fin
```

Maintenant que nous avons décrit le fonctionnement général de la méthode nous présentons l'algorithme d'insertion d'une mesure dans la structure.

Le mécanisme d'insertion d'une mesure dans le cube est très similaire à celui décrit dans la section 3.1.3 pour une dimension. L'algorithme 2 formalise les principales étapes. Si la cellule n'existe pas, un constructeur est appelé (ligne 3). Celui-ci a pour rôle de créer la structure associée et d'insérer la mesure. Dans le cas contraire, il faut vérifier si la cellule n'a pas été mise à jour au cours du même batch (le même n-uplet apparaît plusieurs fois dans le batch). Dans ce cas, il faut simplement ajouter mes au début de l'intervalle correspondant (ligne 6). Sinon, une insertion *classique* s'opère. Si l'intervalle n'est pas plein (ligne 8), cela signifie que l'on peut insérer la mesure sans avoir à changer de fenêtre. L'opération consiste alors à insérer mes au début de l'intervalle (entraînant un décalage des autres mesures de l'intervalle). Enfin, s'il faut changer de fenêtre, deux cas apparaissent :

- Si l'intervalle suivant est stocké dans la même cellule (ligne 12), alors il y a agrégation de l'intervalle courant selon la fonction choisie (ligne 13), insertion de la mesure au début de l'intervalle et appel récursif de insèreValeur avec les paramètres de la ligne 16
- Sinon, on suit la même démarche à la différence que l'appel récursif ne se fait pas sur $cell$ mais sur la généralisation de $cell$ stockant l'intervalle supérieur (ligne 20)

4. EXPÉRIMENTATIONS

4.1. Protocole d'évaluation

Nous évaluons les limites de notre approche en réalisant une étude approfondie de ses performances sur des jeux de données synthétiques et réels. Pour ce faire, nous testons séparemment l'influence de plusieurs paramètres (nombre de dimensions, degré et profondeur des hiérarchies et fonctions de précision). Comme nous l'avons vu précédemment, deux conditions sont indispensables pour valider l'applicabilité de l'approche dans le cas des flots :

1. borner la mémoire utilisée. Cette condition

Algorithme 2 : insèreValeur

Données : $cell$ l'identifiant de la cellule où insérer la mesure, mes la mesure à insérer, $window$ l'intervalle de la *tilted-time window* où il faut insérer mes, $batch$ le numéro du batch

```
1  si (∄ cell) alors
2      newCellule(cell,mes,batch) ;
3  sinon
4      si (LastMAJ(cell) = batch) alors
5          cell.window[0]+=mes ;
6      sinon
7          si ( !full(cell.window)) alors
8              unshift(cell.intervalle,mes);
9              LastMAJ(cell) = numBatch ;
10         sinon
11             si here(cell,window + 1) alors
12                 agr=agregation(cell.window);
13                 cell.window[0]=mes ;
14                 LastMAJ(cell) = batch ;
15                 insereValeur(cell,agr,window+
                   1,batch) ;
16             else
17                 agr=agregation(cell.window);
18                 cell.window[0]=mes ;
19                 cellUp=up(cell,window + 1);
20                 LastMAJ(cell) = batch ;
21                 insereValeur(cellUp,agr,window+
                   1,batch) ;
```

est respectée lors de nos expérimentations car la mémoire nécessaire est toujours inférieure à 10M.

2. borner le temps de calcul entre chaque batch. Cette condition est aussi respectée car pour chaque étape, l'implémentation de notre approche ne dépasse pas la taille du batch.

Nous présentons ici nos résultats obtenus sur des données synthétiques. Le générateur de batchs de données multidimensionnelles utilisé produit aléatoirement des n-uplets satisfaisant certaines contraintes (nombre de dimension, degré des hierarchies...). Nous pouvons ainsi évaluer précisemment notre approche. La convention pour nommer les jeux de données est la suivante : D4L3C5B100T20 signifie qu'il y a 4 dimensions avec 3 niveaux de précision chacune, que les arbres des hiérarchies sont de degré 5 et que les tests ont été réalisés sur 100 batchs de 20K n-uplets. Toutes les expérimentations ont été exécutées sur un Pentium Dual Core 3GHz avec 2Go de mémoire vive sous un Linux 2.6.20. Les méthodes ont été implémentées en Perl version 5 et utilisent une base de données MySQL. Notons que nous utilisons une fenêtre temporelle logarithmique de taille 8 ($log_2(250) = 7,96...$).

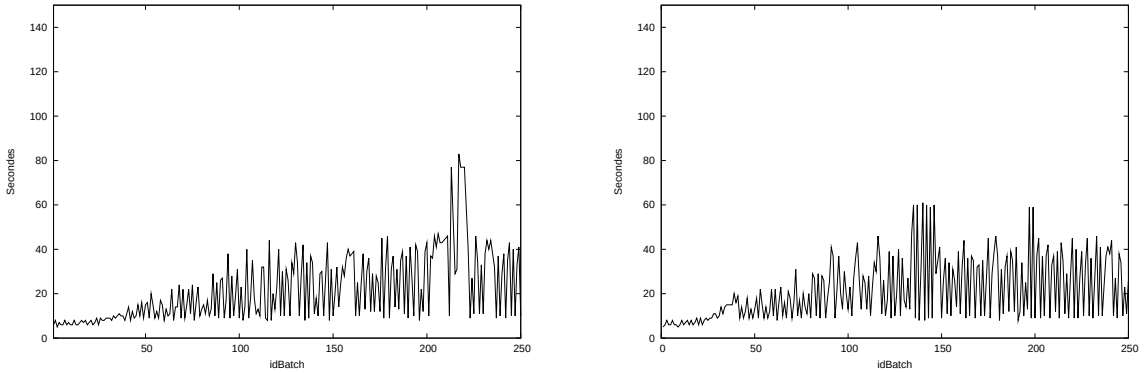
Nous poursuivons avec l'application de la proposition sur un jeu de données réel issu de sondes réseau. Le jeu de test contient 200 batchs de 20 K n-uplets chacun. Un n-uplet est décrit sur 13 dimensions. La moitié sont hiérarchisées (profondeur 2 ou 3). Les résultats obtenus sur ce jeu de données sont meilleurs et permettent de conclure sur la faisabilité de notre approche.

4.2. Résultats sur des jeux de données synthétiques

Le rôle essentiel des fonctions de précision dans le coût de matérialisation a déjà été évoqué dans la section 3.1.5. Dans cette première serie d'expérimentations nous étudions cette fois-ci leur impact sur les performances de notre proposition. Les configurations testées sont présentées dans le tableau 3. La première (*expensive*) conserve l'essentiel de l'historique sur le cuboïde le plus précis. A l'inverse, *Slim* décrit une configuration conservant la majorité de l'historique au plus haut niveau d'abstraction. Enfin, la configuration *Balanced* est une configuration un peu plus réaliste car (1) les fonctions ne sont pas identiques et (2) la plupart des intervalles ne sont pas matérialisés sur le même cuboïde. Les autres caractéristiques du jeu de données sont D5L3C5B250T20.
Les figures 13 et 14 présentent les résultats obtenus. Les temps de traitement sont exposés dans les

Intervalles	***Expensive***	***Slim***	***Balanced***
$[t_0, t_1]$	$A_3B_3C_3D_3E_3$	$A_3B_3C_3D_3E_3$	$A_3B_3C_3D_3E_3$
$[t_1, t_2]$			
$[t_2, t_3]$			$A_3B_2C_2D_3E_3$
$[t_3, t_4]$			$A_2B_2C_2D_3E_2$
$[t_4, t_5]$		$A_1B_1C_1D_1E_1$	$A_1B_2C_2D_3E_1$
$[t_5, t_6]$			$A_1B_1C_1D_2E_1$
$[t_6, t_7]$			$A_1B_1C_1D_1E_1$
$[t_7, t_8]$	$A_1B_1C_1D_1E_1$		

Table 3: Cuboïdes matérialisés



(a): Temps vs *Expensive* (D5L3C5B250T20)

(b): Temps vs *Slim* (D5L3C5B250T20)

Fig. 13: Influence des fonctions de précision (1/2)

figures 13(b), 13(a) et 14(a). Les courbes obtenues suivent un comportement similaire. Premièrement, le temps de traitement augmente légèrement pendant les premiers batchs à cause du faible nombre de cellules présentes dans la structure. Nous observons ensuite une alternance entre des temps très rapides et et d'autres plus longs. Les pics correspondent aux batchs où un changement de fenêtre s'effectue. Enfin, le temps maximal est borné à environ 60 secondes. Nous notons également que l'impact des fonctions de précision est minime sur l'aspect temporel de notre évaluation. Concernant la mémoire vive consommée, celle-ci est présentée dans la figure 14(b). Nous ne présentons ici qu'une seule courbe pour les trois tests car celles-ci sont identiques et constantes.

Dans un contexte de flot multidimensionnel, il est naturel de tester les limites de notre approche sur un nombre variable de dimensions. Le jeu de tests utilisés est L3C5B250T20. Les fonctions de précisions utilisées sont équilibrées (proches de la configuration *Balanced*). Les figures 15 et 16 présentent les résultats obtenus sur l'aspect spatial. Ces courbes présentent les mêmes caractéristiques que lors de la première série de tests. De plus, même si les performances se dégradent avec l'augmentation du nombre de dimensions, le temps de traitement reste borné. Nous ne présentons pas ici de courbe sur la mémoire vive consommée car les résultats sont identiques à la première série.

L'influence du nombre de dimensions est plus important que celle des fonctions de précision mais reste malgré tout limitée. De plus, même avec 15 dimensions, le temps de traitement ne dépasse pas une minute.

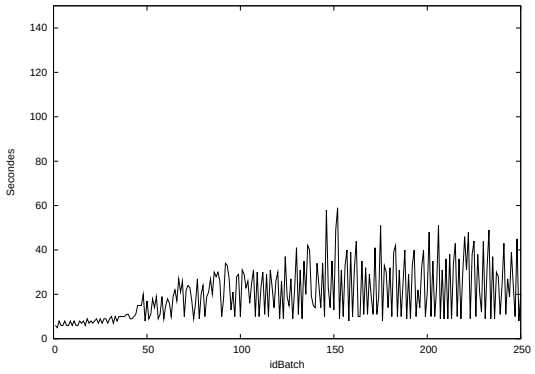
Le degré des hiérarchies détermine le nombre de n-uplets pouvant apparaître dans le flot. Puisque celui-ci influe directement sur le nombre de cellules de notre structure, nous faisons varier ce paramètre pour mesurer son impact sur la proposition. Le jeu de données utilisé est D5L3B250T20. Les figures 17 et 18 montrent des résultats très semblables à la précédente série de tests. La consommation de mémoire étant identique aux autres séries de tests, nous ne les présentons pas. Une fois de plus, les résultats obtenus témoignent des bonnes propriétés du maintien de notre structure.

Nous testons maintenant l'influence de la profondeur des hiérarchies qui déterminent le nombre de cuboïdes matérialisés et influent sur le nombre de généralisation effectuées pour passer d'une fenêtre à une autre. Nous cherchons à déterminer ici quel est l'impact de ces généralisations. Le jeu de test utilisé est D5C5B250T20. Les résultas sont présentés dans les figures 19 et 20. Le temps de traitement suit le même comportement que pour les autres expérimentations. La consommation de mémoire est encore stable à 10Mo.

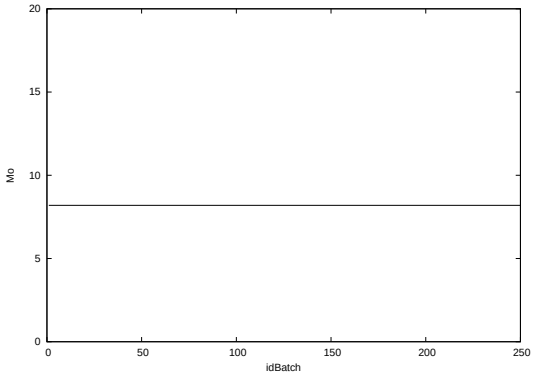
Pour conclure ces séries de tests sur des données synthétiques nous dressons quelques conclusions : malgré de grandes variations dans nos jeux de données, (1) la consommation de mémoire vive reste stable et très faible et (2) le temps de traitement maximal par batch se stabilise après une cinquantaine de batchs pour atteindre une valeur très acceptable. Ces bons résultats sur les deux aspects critiques dans les flots témoignent des caractéristiques intéressantes de notre proposition.

4.3. Application sur des données réelles

La figure 21 expose les résultats sur le jeu de données réels décrit dans la sous-section 4.1. Ce test obtient de meilleurs résultats que sur les jeux de données synthétiques d'un point de vue temps de traitement (21(a)). En effet, celui-ci reste borné à 15 secondes. Ces performances s'expliquent par le nombre plus faible de tuples pouvant circuler dans le flot. Ainsi, les étapes de changement de fenêtre, généralisation et mise à jour sont moins coûteuses. Notons également que, comme pour les données synthétiques, les cinquante premiers batchs sont traités plus rapidement. Enfin, la consommation de mémoire présentée dans la figure 21(b) reste stable et faible tout comme dans les précédentes expérimentations.

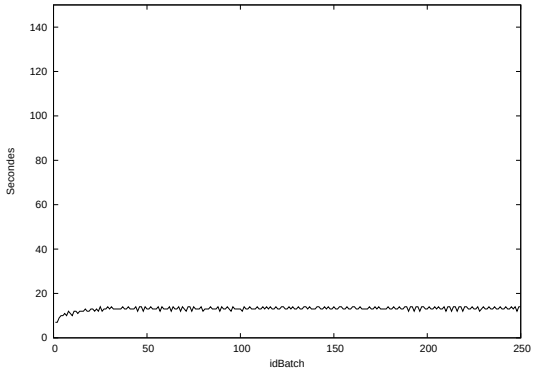


(a): Temps vs *Balanced* (D5L3C5B250T20)

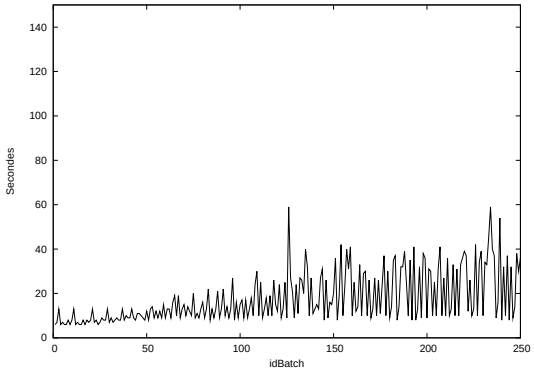


(b): Mémoire vive consommée (D5L3C5B250T20)

Fig. 14: Influence des fonctions de précision (2/2)

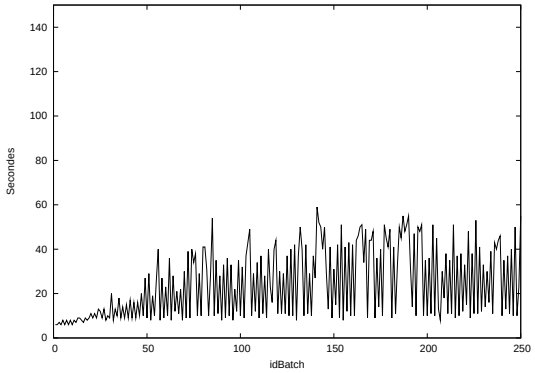


(a): Temps vs 2 dimensions (L3C5B250T20)

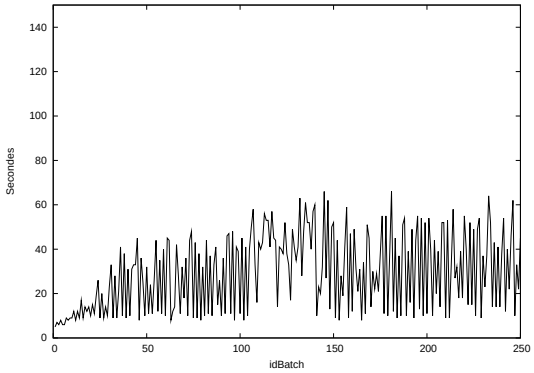


(b): Temps vs 5 dimensions (L3C5B250T20)

Fig. 15: Influence du nombre de dimensions (1/2)

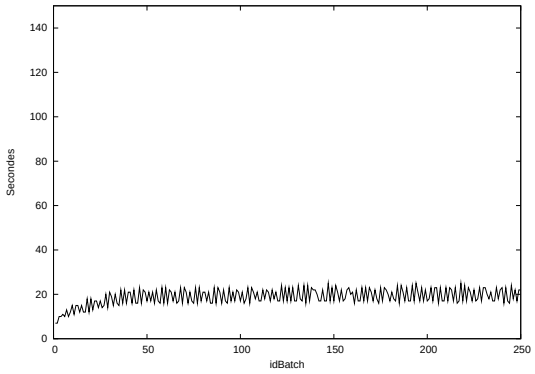


(a): Temps vs 10 dimensions (L3C5B250T20)

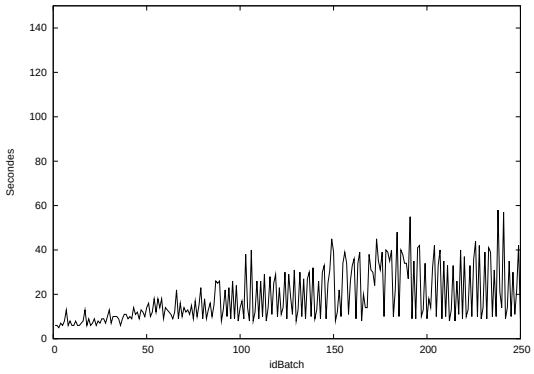


(b): Temps vs 15 dimensions (L3C5B250T20)

Fig. 16: Influence du nombre de dimensions (2/2)

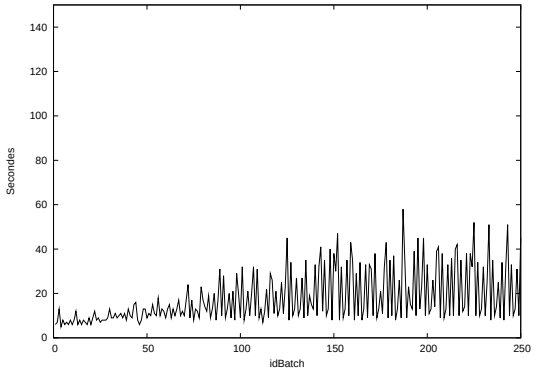


(a): Temps vs degré 2 (D5L3B250T20)

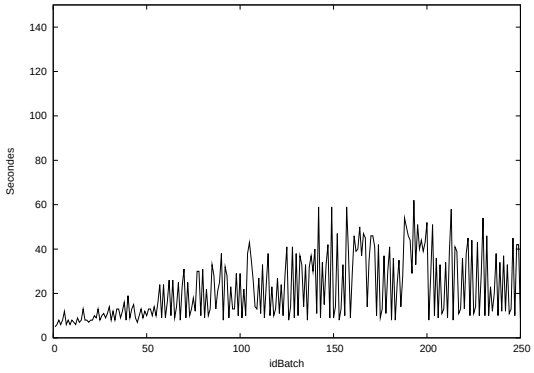


(b): Temps vs degré 5 (D5L3B250T20)

Fig. 17: Influence du degré des hiérarchies (1/2)

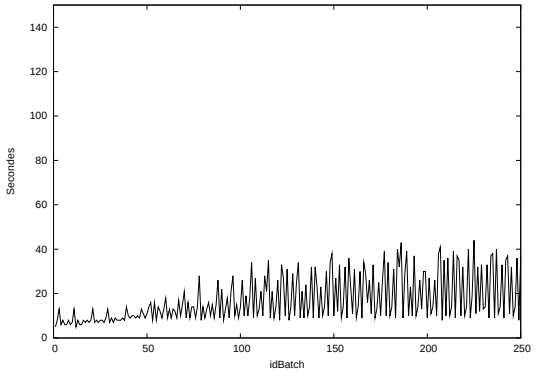


(a): Temps vs degré 10 (D5L3B250T20)

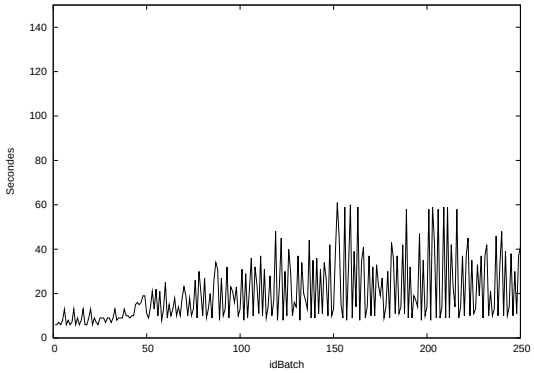


(b): Temps vs degré 20 (D5L3B250T20)

Fig. 18: Influence du degré des hiérarchies (2/2)

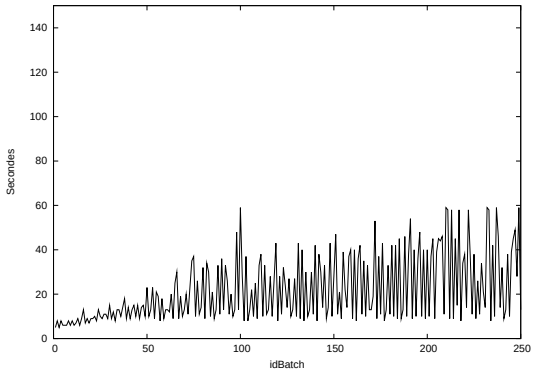


(a): Temps vs profondeur 3 (D5C5B250T20)

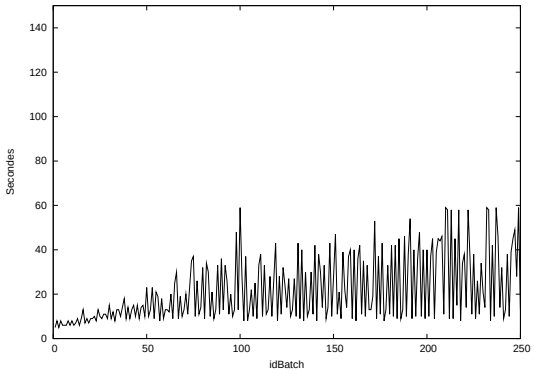


(b): Temps vs profondeur 4 (D5C5B250T20)

Fig. 19: Influence de la profondeur des hiérarchies (1/2)

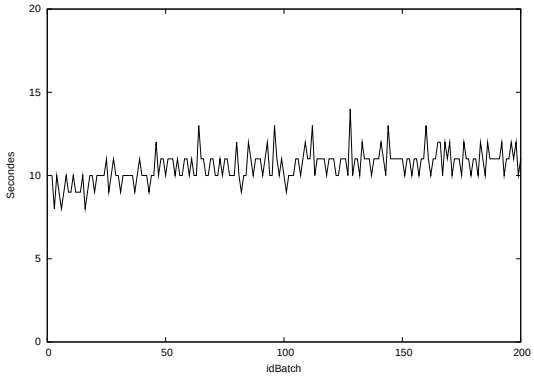


(a): Temps vs profondeur 5 (D5C5B250T20)

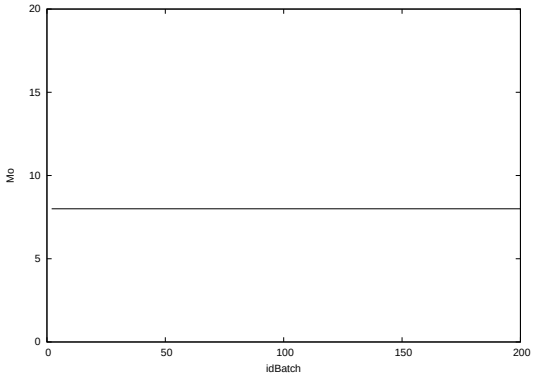


(b): Temps vs profondeur 7 (D5C5B250T20)

Fig. 20: Influence de la profondeur des hiérarchies (2/2)



(a): Temps



(b): Mémoire

Fig. 21: Résultats obtenus sur des données réelles

5. CONCLUSION

Dans cet article, nous nous intéressons à l'adaption des techniques OLAP au contexte des flots. Pour ce faire, nous présentons une architecture compacte et qui permet une analyse multidimensionnelle et multi-niveaux. Puisque habituellement l'historique ancien d'un flot n'est pas consulté précisement, nous avons étendu le principe des *tilted-time window* à toutes les dimensions hiérarchisées. En effet, les fonctions de précision permettent de définir le moment où un niveau de granularité n'est plus consulté. La combinaison de ces fonctions conduit à une structure extrèmement compacte mais capable de répondre à la plupart des requêtes habituelles des utilisateurs. Les expérimentations menées sur des jeux de données synthétiques et réels obtiennent de très bons résultats sur les deux aspects critiques dans le contexte de flot : le temps d'insertion et l'espace occupé par la structure. Ces bonnes performances nous autorisent à envisager de nombreuses perspectives intéressantes. Pour conclure, nous en présentons quelques unes.

Premièrement, l'interrogation d'une telle structure n'est pas triviale. En effet, même si notre méthode minimise l'imprécision générée, nous devons en tenir compte. Pour cela, une piste possible est d'étendre les travaux de [15] pour les adapter à notre contexte.

Enfin, la simple interrogation de la structure n'est pas une tâche suffisante dans la mesure où de nombreuses applications demandent des techniques de fouilles plus sophistiquées (clustering, recherche d'anomalies, de motifs fréquents...). Une perspec-

tive prometteuse est donc d'intégrer de telles techniques à l'aide, par exemple, de fonctions d'agrégations plus complexes et adaptées.

6. RÉFÉRENCES

[1] S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. In *Proc. 22nd Int. Conf. Very Large Databases, VLDB*, pages 506–521, 3–6 1996.

[2] A. Arasu, S. Babu, and J. Widom. The cql continuous query language : semantic foundations and query execution. *The VLDB Journal*, 15(2) :121–142, 2006.

[3] M. Charikar, L. O'Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems, 2003.

[4] Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis, editors. *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*. ACM, 2006.

[5] eBay. 2006 annual report, 2006.

[6] C. Giannella, J. Han, J. Pei, X. Yan, and P. Yu. Mining frequent patterns in data streams at multiple time granularities, 2002.

[7] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube : A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *J. Data Mining and Knowledge Discovery*, 1(1) :29–53, 1997.

[8] J. Han, Y. Chen, G. Dong, J. Pei, B. W. Wah, J. Wang, and Y. D. Cai. Stream cube : An architecture for multi-dimensional analysis of data streams. *Distrib. Parallel Databases*, 18(2), 2005.

[9] Hyo-Sang Lim, Jae-Gil Lee, Min-Jae Lee, Kyu-Young Whang, and Il-Yeol Song. Continuous query processing in data streams using duality of data and queries. In Chaudhuri et al. [4], pages 313–324.

[10] M. Motoyoshi, T. Miura, and I. Shioya. Clustering stream data by regression analysis. In *ACSW Frontiers '04 : Proceedings of the second workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation*, pages 115–120, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.

[11] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Singh Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, approximation, and resource management in a data stream management system. In *CIDR*, 2003.

[12] M. Muthukrishnan. Tutorial of muthu muthukrishnan (rutgers univ.), news february 19th 2007, 2007.

[13] S. Muthukrishnan. Data streams : algorithms and applications. In *SODA '03 : Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 413–413, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[14] Spiros Papadimitriou and Philip S. Yu. Optimal multi-scale patterns in time series streams. In Chaudhuri et al. [4], pages 647–658.

[15] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. Supporting imprecision in multidimensional databases using granularities. In *SSDBM '99 : Proceedings of the 11th International Conference on Scientific on Scientific and Statistical Database Management*, page 90, Washington, DC, USA, 1999. IEEE Computer Society.

[16] C. Raïssi and P. Poncelet. Sampling for sequential pattern mining : From static databases to data streams. In *ICDM*, pages 631–636, 2007.