

UNIVERSITÉ MONTPELLIER 2
— SCIENCES ET TECHNIQUES —

THÈSE

pour obtenir le grade de
Docteur de l'Université Montpellier 2

DISCIPLINE : INFORMATIQUE
Spécialité Doctorale : *Informatique*
Ecole Doctorale : *Information, Structure, Systèmes*

présentée et soutenue publiquement par

Yoann PITARCH

le 10 mai 2011

Résumé de Flots de Données : Motifs, Cubes et Hiérarchies

JURY

Joao GAMA, Professeur, University of Porto, Portugal, Rapporteur
Olivier TESTE, Maître de Conférences, HDR, Université Paul Sabatier, Toulouse, Rapporteur
Christine COLLET, Professeur, INP Grenoble, Examineur
Cécile FAVRE, Maître de Conférences, Université Lyon 2, Examineur
Torben Bach PEDERSEN, Professeur, Aalborg University, Danemark, Examineur
Patrick VALDURIEZ, Directeur de Recherche, INRIA, Examineur
Anne LAURENT, Maître de Conférences, HDR, Université Montpellier 2, Co-directrice de thèse
Pascal PONCELET, Professeur, Université Montpellier 2, Directeur de thèse

Remerciements

Une thèse est le résultat d'un travail de longue haleine. C'est le fruit de multiples discussions au cours de diverses collaborations. L'organisation de ces idées qui constitue ma contribution modeste mais, je l'espère, originale, n'est pas possible sans un cadre de travail matériel et intellectuel favorable.

C'est pourquoi je tiens à remercier chaleureusement mes directeurs de thèse Anne Laurent et ¹ Pascal Poncelet qui m'ont donné d'excellentes conditions de travail. Qu'ils reçoivent toute ma gratitude pour m'avoir laissé si souvent le champ libre et m'avoir fait confiance dans mes choix et mes entêtements tout en m'apportant leurs rigueurs scientifiques. Mes premiers écrits se rappellent encore d'un certain stylo rouge qui devint ensuite noir. Pour cela et pour tout le reste, je leur suis extrêmement reconnaissant.

Je souhaite également remercier Joao Gama et Olivier Teste pour m'avoir fait l'immense honneur d'accepter d'être les rapporteurs de ce mémoire et pour le temps qu'ils ont consacré à cette tâche. Je remercie également Christine Collet, Cécile Favre, Patrick Valduriez et Torben Bach Pedersen pour avoir accepté de participer à l'évaluation de ce travail par leur présence lors de la soutenance de ce mémoire.

J'ai eu également plaisir à discuter avec d'autres chercheurs qui ne font pas partie des fonctions officielles de cette thèse. Merci donc à Roland Ducourneau pour avoir participé au suivi de cette thèse au travers du comité de suivi de thèse. Son expérience m'a été fort utile. Merci à Maguelonne Teisseire qui a su me conseiller jusqu'au dernier et ultime moment. Je n'oublie pas les membres de l'équipe fouille de données du LIRMM pas plus que les chercheurs qui gravitent autour de celle-ci.

Pendant, ces trois années, deux doctorants ont eu la (mal)chance de partager mon bureau. Merci donc à Paola Salle pour avoir supporté mon aptitude à parler tout seul ainsi qu'à Julien Rabatel pour être resté stoïque pendant la redoutable phase de rédaction. Puisse le meilleur vous arriver.

Je n'oublie pas mes prédécesseurs, Marc, Chedy, Lisa, Cécile et tous les autres qui ont su m'ouvrir la voie et dont les conseils furent très précieux dans ce long cheminement que représente une thèse. Une pensée également pour ceux qui suivront Julien, Hassan, Pu, Laurent, Mickael, etc.

1. Dans cette page de remerciements, le « *et* » est un opérateur commutatif.

Merci aux membres du Département Informatique de l'IUT de Montpellier pour leur accueil, leur soutien et leur confiance renouvelée lors de mes trois années de monitorat.

Je remercie ma famille et mes amis qui m'ont soutenu patiemment pendant cette période. Vous avez inconsciemment joué un grand rôle dans la réussite de cette longue épreuve.

Enfin, *last but not least*, merci à Adeline d'avoir su m'accompagner et m'encourager depuis le début et ceci malgré mon manque de disponibilité.

*A ceux qui ont rendu cette aventure possible
Trouvez dans ce travail l'expression de ma plus profonde gratitude*

Là où la volonté est grande, les difficultés diminuent.

NICOLAS MACHIAVEL

Sommaire

Remerciements	3
Sommaire	7
1 Introduction	13
Introduction	13
1 Contexte général	13
2 Cas d'étude	17
3 Principales contributions	18
4 Organisation du mémoire	21
2 État de l'art des approches de résumés de flots	23
1 Construction de cubes de données	23
1.1 Définitions préliminaires	23
1.2 Bases de données multidimensionnelles	26
2 Manipuler des flots de données	29
2.1 Modèles de flots	29
2.2 Gestion du temps dans les flots	30
3 Panorama des techniques de résumé de flots de données	33
3.1 Utilisation de méthodes statistiques	33
3.2 Utilisation de techniques de fouille de données	34
3.3 Utilisation de techniques de construction de cubes de données	37
4 Données manipulées	38
5 Discussion	39
5.1 Critères d'évaluation	40

5.2	Discussion des approches présentées	41
3	Window Cube, une méthode de construction de cube à partir d'un flot de données	47
1	Introduction	47
2	Window Cube	48
2.1	Aperçu général de l'approche	48
2.2	Formalisation	49
2.3	Mise à jour	52
2.4	Algorithmes	54
3	Automatiser la définition des fonctions de précision	55
3.1	Aperçu général	56
3.2	Formalisation	58
4	Interroger Window Cube	61
4.1	Requêtes considérées et satisfaisabilité	61
4.2	Gérer l'imprécision	63
5	Expérimentations	65
5.1	Comparaison avec Stream Cube	66
5.2	Résultats sur des jeux de données synthétiques	68
5.3	Application sur des données réelles	72
6	Discussion	73
4	Prise en compte de la distribution variable des données dans un flot	79
1	Introduction	79
2	Aperçu général de l'approche	81
3	Listes dynamiques : définitions et fonctionnement	81
3.1	Définitions	82
3.2	Fonctionnement des listes dynamiques	83
4	Résumer un flot de données multidimensionnelles brutes	85
4.1	Mise à jour de la structure	86
5	Résumer des itemsets fréquents	88
5.1	Motivations	88
5.2	Extraction d'itemsets multidimensionnels fréquents dans des batchs	89

5.3	Principe général	89
5.4	Adaptation de la structure existante	91
6	Expérimentations	92
6.1	Validation expérimentale du passage à l'échelle	93
6.2	Synthèse d'itemsets fréquents	98
6.3	Représentativité du résumé produit	99
7	Discussion	100
5	Des séquences fréquentes pour résumer un flot de données	103
1	Introduction	103
2	Concepts préliminaires et notations	104
2.1	Problématique de l'extraction des motifs séquentiels multidimensionnels . .	104
2.2	Un automate indexant les séquences	106
3	SALINES : un algorithme d'extraction de motifs multiniveaux	109
3.1	Aperçu général de l'approche	109
3.2	Etat de l'art des algorithmes d'extraction de motifs multidimensionnels multiniveaux	109
3.3	Phase 1 : construction de l'automate des sous-séquences	110
3.4	Phase 2 : à la recherche des motifs multiniveaux	113
3.5	Extension aux données multidimensionnelles	118
3.6	Expérimentations	119
3.7	Résultats sur un jeu de données réelles	121
3.8	Premières conclusions	122
4	ALIZES : des motifs séquentiels pour résumer un flot de données	123
4.1	Aperçu général de l'approche	123
4.2	Extraction des fréquents et obtention des fermés	124
4.3	Union d'automates et réduction	127
4.4	Extension aux données multidimensionnelles	132
4.5	Un processus parallélisable	132
4.6	Expérimentations	133
4.7	Conclusions sur ALIZES	136
5	Discussion	137

6	Un modèle conceptuel pour les entrepôts de données contextuels	139
1	Introduction	139
2	Cas d'étude	141
3	Panorama des travaux existants	144
3.1	Panorama des types de hiérarchies	144
3.2	Panorama des modèles conceptuels d'entrepôts de données	147
3.3	Panorama des approches introduisant de la flexibilité dans l'analyse des données	149
3.4	Panorama des méthodes de conception du schéma d'un entrepôt	150
3.5	Discussion	151
4	Modélisation conceptuelle et graphique	152
5	Exploitation des entrepôts contextuels	159
5.1	Définitions préliminaires	160
5.2	Généralisation contextuelle	161
5.3	Spécialisation contextuelle	162
6	Mise en œuvre	163
6.1	Stockage des connaissances : utilisation d'une base de connaissance externe	163
6.2	Gérer les connaissances expertes	165
7	Discussion	167
7	Implantation dans un démonstrateur	171
1	Introduction	171
2	Architecture de la plateforme MIDAS	172
2.1	Vue d'ensemble de l'architecture de la plateforme MIDAS	172
2.2	Fichiers XML d'entrée	173
2.3	Gestion des tables de la base de données pour le stockage des résumés . . .	178
3	Description de l'outil de visualisation	179
3.1	Menu principal	180
3.2	Choix du flot à afficher	180
3.3	Sélection du type de graphique à utiliser	180
3.4	Lancement de la visualisation	181
4	Approches implantées et besoins applicatifs associés	182

<i>SOMMAIRE</i>	11
4.1 Analyse d'usages de clients d'Orange	182
4.2 Résumé de flots de données de capteurs	183
5 Discussion	184
8 Bilan général et perspectives	185
Bibliographie	191
Publications dans le cadre de cette thèse	201
Annexe	i
A Survol des DSMS existants	iii
A.1 DSMS spécifiques	iii
A.2 DSMS généralistes	vi
Résumé	ix

Chapitre 1

Introduction

1 Contexte général

Notre société est définitivement entrée de plein pied dans l'ère de l'analyse des données. Longtemps utilisées dans les seules grandes entreprises et organisations institutionnelles et scientifiques, l'analyse de données massives est maintenant présente dans de nombreux domaines. Par exemple, dans le domaine de la santé, de plus en plus d'outils (*e.g.* capteurs, puces ADN) permettent de récolter des données qui peuvent se révéler essentielles dans la compréhension et la prévention de certaines maladies.

Pour gérer au mieux ces données, les systèmes de gestion de bases de données (SGBD) basés sur le modèle OLTP (*On Line Transaction Processing*) sont apparus. L'objectif de ce modèle est de permettre l'insertion et la modification de façon rapide et sûre. Les SGBD sont particulièrement adaptés pour répondre efficacement à des requêtes concernant une petite partie des données stockées dans la base. En ce sens, il est d'usage de dire que les SGBD permettent un traitement élémentaire des données.

Alors que le modèle OLTP est particulièrement efficace pour récolter et consolider les informations produites (*e.g.* bases de production), ce modèle transactionnel montre ses limites quant à l'analyse efficace et globale des données stockées. Les bases de données multidimensionnelles se sont naturellement imposées dans le monde industriel car elles pallient les limites des bases de données relationnelles. En effet, dans un contexte d'analyse décisionnelle, l'utilisateur a besoin d'informations agrégées, résumées et observables sur plusieurs niveaux de précision. L'intérêt des décideurs ne se situe alors plus dans l'analyse détaillée des individus mais plutôt dans la recherche de tendances générales concernant un ensemble d'individus. Dès lors, un modèle de traitement OLTP pour réaliser de telles opérations est inadéquat car de nombreuses et coûteuses jointures

seraient nécessaires et dégraderaient les performances du système. Pour répondre à ce besoin et permettre des réponses efficaces à des requêtes complexes posées sur un grand volume de données, les entrepôts de données sont apparus. Une définition de ces entrepôts est donnée dans [Inm96] : *A datawarehouse is a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management decision making progress.* Un entrepôt est donc un ensemble de données organisées autour de la même thématique dans un objectif de prise de décision. Alors, pour analyser les données d'un entrepôt et faciliter le processus, un ensemble d'applications a été proposé pour traiter des données multidimensionnelles : les outils OLAP (*On-Line Analytical Processing*).

Les bases de données multidimensionnelles permettent donc d'analyser, de synthétiser les informations et représentent à ce titre une alternative intéressante aux systèmes relationnels classiques. Les avantages des bases de données multidimensionnelles sont les suivants :

- Gestion optimisée des agrégats,
- Navigation aisée le long des hiérarchies des dimensions,
- Visualisation intuitive des données.

Le tableau 1.1 récapitule les principales différences entre les modèles OLTP et OLAP.

	OLTP	OLAP
Utilisation	SGBD	Entrepôt de données
Opération typique	Mise à jour	Analyse
Type d'accès	Lecture écriture	Lecture
Niveau d'analyse	Elémentaire	Global
Quantité d'information échangées	Faible	Importante
Taille de la base	Faible (quelques Go)	Importante (plusieurs To)
Ancienneté des données	Récente	Historique

TABLE 1.1 – Tableau récapitulatif des différences entre les modèles OLTP et OLAP [Cod70]

Cependant, permettre un stockage efficace et permanent des données via un SGBD ou un entrepôt de données n'est pas suffisant. En effet, face à un tel volume de données, les analyser manuellement peut rapidement devenir fastidieux voire impossible et a contribué à l'essor de la fouille de données (*data mining* en anglais).

Jusqu'ici, nous supposons que l'intégralité des données pouvaient être stockées et analysées "à la demande". Cette supposition est de moins en moins valide et s'explique principalement par l'essor des technologies de communication. Dès lors, face à l'explosion du volume de données transmises, l'augmentation des capacités de stockage apparaît ne plus suffire pour stocker l'intégralité

des données produites. Ainsi, de plus en plus de données circulent sous la forme de *flots de données* de façon continue, à un rythme rapide et éventuellement de manière infinie. Ces flots de données interviennent dans des domaines aussi variés que le trafic réseau, les transactions financières en ligne, les réseaux de capteurs, les flots textuels (dépêches AFP, flux RSS), . . .

Les données disponibles sous forme de flots se caractérisent par leur volumétrie et leur débit qui rendent difficile, voire impossible, leur stockage. La gestion et l'interrogation doivent donc se faire à la volée (*i.e.*, sans stockage préalable). Dès lors, un pan de recherche s'est intéressé à l'interrogation, à l'analyse et à la fouille de données disponibles sous forme de flots. En effet, les approches efficaces sur des données "*statiques*¹" doivent être adaptées pour répondre aux particularités des flots. Principalement, ces approches doivent réunir deux conditions pour pouvoir être appliquées sur un flot de données :

- Vitesse : une des principales caractéristiques des flots est la vitesse à laquelle les données y circulent. Cette spécificité oblige les algorithmes portant sur les flots à respecter deux contraintes. Premièrement, le temps de traitement d'une valeur (ou d'un groupe de valeurs) du flot doit être plus rapide que l'intervalle séparant l'arrivée de deux valeurs (ou groupes de valeurs). Cette condition est nécessaire pour ne pas bloquer le flot. De plus, puisque le débit d'un flot peut être élevé, chaque élément du flot ne peut pas être observé plusieurs fois. Ainsi, un traitement *à la volée* (*i.e.* une seule lecture) des valeurs du flot est indispensable.
- Mémoire : un flot étant potentiellement infini, le stocker intégralement ou en grande partie pour l'analyser est impossible. Dès lors, les approches opérant sur les flots doivent donc nécessairement prendre en compte cet aspect et ne pas nécessiter une trop grande occupation mémoire. Ainsi, il devient de plus en plus acceptable [Mut05] de fournir à l'utilisateur une réponse approximative quand l'analyse demandée réclame de considérer une partie trop importante du flot de données.

De la même manière qu'un SGBD permet d'interroger une base de données, les Systèmes de Gestion de Flot de Données (ou *Data Stream Management System* en anglais) permettent de formuler des requêtes sur un flot de données. La caractéristique principale d'un DSMS est sa capacité à effectuer des requêtes continues sur le flot de données. Une requête conventionnelle sur une base de données statique s'exécute une fois et retourne un ensemble de données à un instant précis. Par contraste, une requête continue s'exécute continûment sur le flot de données tant qu'il est alimenté. Les résultats d'une requête continue sont modifiés à chaque arrivée de données. Concernant les techniques de fouilles sur des données statiques, nombre d'entre elles ont été étendues au contexte

1. Nous appelons données statiques les données stockées dans un SGBD ou un entrepôt par opposition à la dynamique des données d'un flot

dynamique des flots de données.

Malgré le fait que les approches existantes sont très efficaces, une des principales critiques qui peut être formulée à leur encontre est qu’elles exigent de connaître *a priori* les requêtes ou les algorithmes que l’on souhaite appliquer. Cette anticipation des besoins d’analyse peut se révéler critique voire impossible dans certaines situations. Par exemple, considérons un flot de données de capteurs mesurant l’activité d’une centrale nucléaire. Les requêtes usuelles (*e.g.* calcul de valeurs moyennes sur une période de temps) sont bien maîtrisées et peuvent donc être anticipées. Par contre, en cas d’anomalie ou du défaillance du système, le besoin d’analyse sera consacré sur la découverte des causes de cette anomalie et des conséquences éventuellement engendrées. De part le caractère imprévisible d’une anomalie, il n’est pas possible d’anticiper l’exécution de telles requêtes. Dès lors, les données du flot permettant de diagnostiquer la défaillance auront alors été définitivement perdues. Ainsi, permettre la formulation d’une requête ou l’exécution d’un algorithme de fouille de données sur des données anciennes du flot de données apparaît être un enjeu majeur.

Ce nouveau besoin a poussé les chercheurs à s’intéresser à une nouvelle problématique inhérente aux flots de données : “*comment conserver et construire un historique (ou résumé) d’un flot de données afin de permettre son analyse a posteriori ?*”. Cette interrogation soulève quelques caractéristiques qu’un résumé doit nécessairement posséder pour être exploitable :

- Le critère de compacité : par contraste avec un flot de données, son résumé doit être relativement compact et borné en mémoire ;
- Le critère de mise à jour rapide : afin de ne pas être bloquant pour le flot , un résumé de flot de données doit pouvoir être maintenu rapidement ;
- Le critère de représentativité : un résumé doit capturer au mieux l’historique d’un flot de données (*e.g.* cycles périodiques, anomalies) afin que son analyse soit la plus fidèle possible par rapport à l’historique réel.

Au cours de ces dernières années, la communauté s’est donc intéressée à la construction de résumés de données disponibles sous forme de flots. En l’occurrence, le projet MIDAS (ANR-07-MDCO- 008)², dans lequel ce travail s’inscrit, a justement pour vocation de proposer de nouvelles techniques pour résumer des flots de données. Par exemple, certains travaux se sont intéressés à la construction de résumés de flots de données numériques (*i.e.* un flot numérique peut être assimilé à une série temporelle dont la taille est non bornée). Cette problématique étant similaire

2. Ce projet ANR a fait intervenir, entre 2008 et 20011, des partenaires industriels tels que Orange Labs, EDF R&D ou le CHU de Fort de France et des partenaires académiques tels que l’ENST, l’INRIA de Sofia Antipolis, le CERGMIA à Fort de France et le LIRMM à Montpellier

aux travaux menés en théorie du signal, de nombreuses approches se sont appuyées sur des outils mathématiques efficaces et robustes tels que, entre autres, les ondelettes [CF02, PM02], les transformations de Fourier discrètes [FRM94, RM97] ou les transformées en cosinus discrètes pour construire de tels résumés.

Cependant, de plus en plus de données sont symboliques et multidimensionnelles (*e.g.* un appel téléphonique peut être décrit selon plusieurs dimensions telles que l'adresse d'émission, l'adresse de réception, la durée de l'appel, ...). Une spécificité souvent associée aux données multidimensionnelles est la possibilité de les observer sur plusieurs niveaux de granularité. Considérons par exemple un flot représentant les conversations téléphoniques en France. Un des champs (dimensions) composant ce log est la ville d'émission de l'appel. Intuitivement, ce champ peut facilement être considéré selon différents niveaux de granularité tels que le département ou la région. Ces caractéristiques permettent d'envisager l'utilisation de nouvelles techniques de résumé. En particulier, les travaux menés pendant cette thèse se sont intéressés à comment adapter des mécanismes issus des technologies OLAP pour résumer un flot de données multidimensionnelles. Plus particulièrement, nous nous sommes focalisés sur le rôle que les hiérarchies de données peuvent avoir dans notre contexte.

Les travaux présentés dans ce manuscrit s'articulent autour de cette problématique de construction d'un résumé de données multidimensionnelles et hiérarchisées. Ainsi, tout au long de cette thèse, nous nous sommes posés les questions suivantes :

- *Est-il possible de construire un cube de données alimenté par un flot de données satisfaisant les trois critères mentionnés précédemment ?*
- *A quel point les hiérarchies associées aux différentes dimensions peuvent-elle être utiles dans un tel contexte de résumé ?*
- *Certaines techniques de fouille de données peuvent-elles être exploitées pour produire un bon résumé (au sens précédemment décrit) ? Dans le cas contraire (i.e. s'il existe des critères non vérifiés), quelles sont les modifications à apporter pour disposer de bons résumés ?*
- *Les différentes représentations de hiérarchies présentées dans la littérature sont-elles bien adaptées à cette tâche ?*

2 Cas d'étude

Nous utiliserons le cas d'étude suivant afin d'illustrer les concepts et approches proposés dans ce manuscrit. Ainsi, nous considérons l'historique des communications écrites numériques pour quelques individus. Le tableau 1.2 de la figure présente un extrait d'un tel historique. Chaque

Date	Id emetteur	Lieu émission	Support
26/01/2011 10h20	1	Louxor	Twitter
26/01/2011 18h00	1	Le Caire	Facebook
26/01/2011 19h00	2	Tunis	Mail
26/01/2011 19h01	3	Paris	SMS
27/01/2011 08h00	4	Paris	Mail
27/01/2011 09h00	4	Paris	SMS
27/01/2011 10h00	5	Tunis	Tweeter
27/01/2011 14h00	6	Montpellier	Myspace
28/01/2011 09h00	2	Tunis	Facebook
28/01/2011 15h00	2	Tunis	Facebook
28/01/2011 16h00	2	Hammamet	Mail
28/01/2011 17h00	3	Paris	Facebook
28/01/2011 22h00	3	Tunis	Twitter
28/01/2011 23h00	3	Hammamet	Mail

TABLE 1.2 – Extrait des communications écrites numériques émises sur une période de trois jours

n -uplet de ce tableau relate l’envoi, par un individu, d’un message numérique à une date donnée, via un vecteur de communication donné depuis un lieu donné. Par exemple, la première ligne du tableau peut être interprétée de la façon suivante : “*Le 26/01/2011 à 10h20, l’individu 1 a envoyé, depuis Louxor, un message Twitter.*”. Les dimensions décrivant le lieu d’émission et le vecteur de communication peuvent être considérés selon différents niveaux de granularité. La figure 1.1 (resp. figure 1.2) présente la hiérarchie des valeurs associée à la dimension *Lieu d’émission* (resp. *Vecteur de communication*). Sous l’hypothèse qu’une jointure existe entre ces différentes sources de données, nous pouvons considérer que ces données peuvent constituer un flot de données multidimensionnelles.

3 Principales contributions

Les principales contributions présentées dans ce mémoire sont décrites ci-dessous.

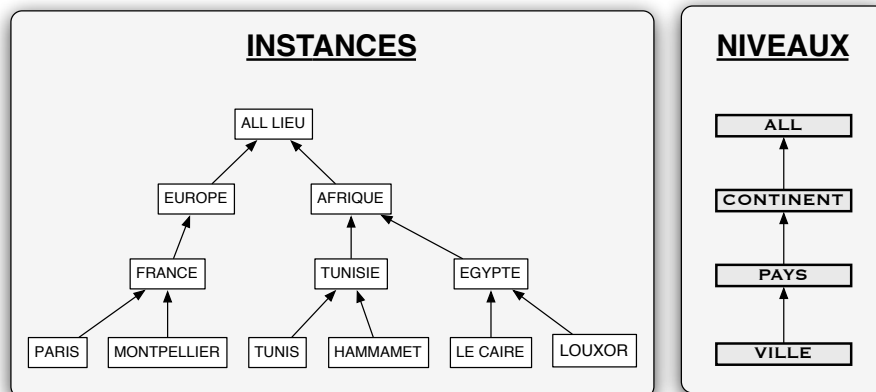


FIGURE 1.1 – Hiérarchie associée à la dimension *Lieu d'émission*

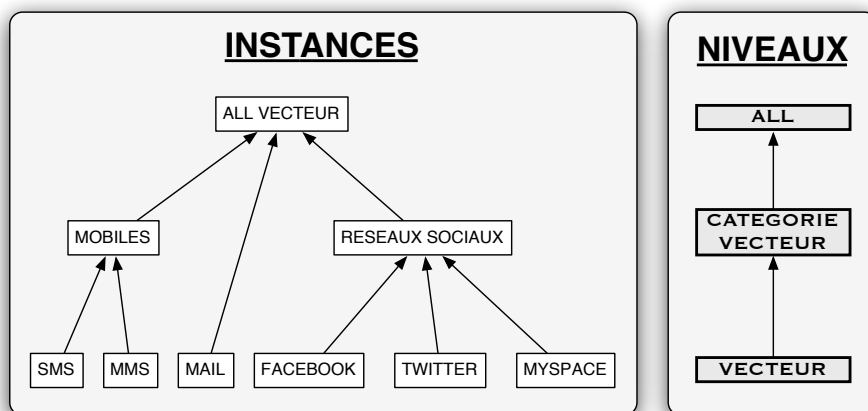


FIGURE 1.2 – Hiérarchie associée à la dimension *Vecteur de communication*

Construction d'un cube de données compact pour résumer un flot de données multidimensionnelles

Nous proposons une méthode de construction d'un cube de données palliant deux limites de Stream Cube [HCD⁺05], la seule approche existant dans la littérature pour répondre à la problématique du résumé de flot de données multidimensionnelles hiérarchisées. En effet, en réduisant à la fois le coût de stockage et le temps de mise à jour de la structure, l'approche proposée répond aux critères de compacité et de mise à jour rapide de la structure. En outre, nous fournissons une méthode automatique pour paramétrer l'approche et proposons un système de réponse aux requêtes adapté à l'imprécision inhérente aux résumés de flots de données.

Retenir précisément les événements atypiques d'un flot de données multidimensionnelles

Les deux approches existantes pour construire un cube de données souffrent d'une limite critique quant à la représentativité de l'historique stocké. En effet, l'utilisation d'un mécanisme d'agrégation systématique de l'historique rend impossible la conservation des apparitions précises d'un élément occasionnel du flot. Nous proposons alors une technique basée sur un graphe et des listes dynamiques pour restituer fidèlement l'historique de tels éléments. Dès lors, l'approche proposée permet de construire un résumé d'un flot de données multidimensionnelles hiérarchisées qui valide le critère de représentativité. S'appuyant sur la même structure de graphe et de listes dynamiques, nous montrons comment une approche d'extraction d'itemsets fréquents peut être considérée comme un *bon* résumé. Le choix de l'utilisation des itemsets fréquents plutôt qu'une autre approche de fouille de données est motivé par sa proximité avec une théorie proposée en psychologie cognitive sur le fonctionnement de la mémoire humaine.

Une nouvelle approche de fouille de données comme technique de résumé

Nous nous intéressons ensuite à l'utilisation des motifs séquentiels pour résumer un flot de données. Pour cela, nous proposons, dans un premier temps, un algorithme d'extraction de motifs séquentiels multidimensionnels et multiniveaux sur des bases statiques permettant la découverte de motifs qui ne pouvaient être extraits par les approches existantes dans la littérature. Basé sur une structure d'automate, cette approche repose sur des mécanismes de fusion d'états pour exhiber des généralisations fréquentes. Nous nous inspirons ensuite des mécanismes proposés dans cet algorithme d'extraction pour proposer une méthode de résumé adaptée aux flots. Dans un premier, des automates indexant les motifs fréquents sont construits à partir du flot. Pour satisfaire le critère de compacité, ces automates sont ensuite fusionnés entre eux et compressés grâce à l'utilisation des hiérarchies associées aux données.

Définition d'un nouveau modèle d'entrepôt autorisant la généralisation contextuelle d'attributs numériques

Au cours de ce travail, nous avons été amenés à mettre en place des hiérarchies à partir de données numériques. Par exemple, lorsque l'on considère une mesure de capteur relevant la tension artérielle d'un patient, une hiérarchie naïve serait de regrouper les valeurs par intervalle fournissant un résumé tel que “entre t_1 et t_2 , la tension artérielle se situait dans l'intervalle $[v_1; v_2]$, puis entre t_2 et t_3 celle-ci se situait entre $[v_2; v_3]$, ...”. Malgré tout, dans un contexte de détection d'anomalies par exemple, il est souvent nécessaire de donner une sémantique à cette information numérique. Par exemple, une tension artérielle peut être qualifiée de *très faible*, *faible*, *normale*, ... permettant ainsi de produire un résumé plus intéressant et plus significatif pour un médecin. Dès lors, considérer des hiérarchies “traditionnelles” pour de telles généralisations sémantiques suppose que ces généralisations sont universelles (*e.g.* une tension à 14 est normale pour tous les patients). Or, une étude du domaine a révélé que cette généralisation est bien souvent contextuelle (*e.g.* une tension normale ne correspond pas à la même mesure pour un enfant ou un adulte). A notre connaissance, aucune représentation formelle de hiérarchie ne permet de représenter de tels contextes pour guider la généralisation. Ainsi, nous proposons une nouvelle catégorie de hiérarchies, les *hiérarchies contextuelles*, pour prendre en compte la connaissance experte dans le mécanisme de généralisation. Nous intégrons ces nouvelles hiérarchies dans un nouveau modèle d'entrepôt de données, les entrepôts contextuels. Ces entrepôts contextuels sont définis formellement grâce aux modèles conceptuel et graphique. Nous proposons également un modèle logique pour la manipulation de ces hiérarchies et leur exploitation au sein d'un entrepôt.

Vers une plateforme d'interrogation générique de résumé de flots de données

L'objectif du projet MIDAS est de proposer une plate-forme intégrant les principales approches proposées par les partenaires académiques pour répondre aux besoins des partenaires industriels. Nous montrons comment deux des approches proposées dans ce manuscrit ont été intégrées à ce démonstrateur et comment ces approches répondent aux besoins formulés par les industriels.

4 Organisation du mémoire

Ce manuscrit s'organise de la façon suivante.

Dans le chapitre 2, nous présentons un panorama des travaux de la littérature connexes à notre problématique. Dans le chapitre 3, nous proposons une méthode pour construire efficacement un cube de données compact alimenté par un flot de données multidimensionnelles. Les structures de données proposées jusqu'ici pour résumer un tel flot de données entraînent une grande perte de

précision quand il s'agit de résumer des éléments apparaissant peu dans le flot, nous proposons une nouvelle méthode palliant cette limite dans le chapitre 4. Dans le chapitre 5, nous nous intéressons à la combinaison entre les motifs séquentiels et les hiérarchies pour résumer un flot de données. Ainsi, nous proposons dans un premier temps *SALINES*, un algorithme d'extraction de motifs séquentiels exploitant plus justement les hiérarchies. Ensuite, nous nous intéressons à l'adaptation de ce nouveau type de connaissance dans un contexte de flot de données et proposons *ALIZÉS*, une méthode de résumé de flot de données. Dans le chapitre 6, nous présentons un nouveau type d'entrepôt pour prendre en compte les hiérarchies contextuelles. Ensuite, nous présentons comment certaines de nos approches ont été intégrées à la plateforme résultant du projet MIDAS. Enfin, nous finissons par un bilan où nous discutons notamment des perspectives associées aux différentes propositions de ce travail.

Chapitre 2

État de l'art des approches de résumés de flots

Notre travail se situe à l'intersection de deux domaines actifs de recherche : les bases de données multidimensionnelles et les flots de données. Nous consacrons donc les deux premières sections de ce chapitre à dresser un panorama des approches existantes dans ces deux domaines. Nous présentons ensuite les principales approches de résumé de flots de données. Enfin, dans la dernière section, nous discutons des travaux présentés dans les précédentes sections.

1 Construction de cubes de données

1.1 Définitions préliminaires

La modélisation multidimensionnelle consiste à représenter les sujets d'analyse appelés *faits* autour d'axes d'analyse appelés *dimensions*. Les faits sont constitués d'indicateurs d'analyse appelés *mesures*. Les dimensions sont composées d'attributs modélisant les différents niveaux de granularité des axes d'analyse. Ces attributs composent ainsi la hiérarchie de la dimension à laquelle ils sont attachés. Nous formalisons maintenant ces concepts en nous basant sur les notations introduites dans [PLT07].

Soit $\mathcal{D} = \{D_1, D_2, \dots, D_n\}$ un ensemble de dimensions où chaque dimension D_i est définie sur un domaine potentiellement infini mais dénombrable de valeurs, noté $dom(D_i)$. Pour chacune des dimensions, nous supposons que $dom(D_i)$ contient une valeur spécifique notée ALL_i pouvant être interprétée comme *toutes les valeurs de D_i* . Nous supposons que chaque dimension D_i est associée à une hiérarchie notée $H_i = (D_i^{max_i}, \dots, D_i^0 = ALL_i)$ où les différents D_i^p désignent les niveaux de granularité de la hiérarchie H_i . Le niveau D_i^p est dit *moins spécifique* que le niveau

$D_i^{p'}$ si $p < p'$. Nous notons cette relation $D_i^p \succ D_i^{p'}$. Chaque hiérarchie H_i est un arbre équilibré de hauteur $\max_i - 1$ dont les nœuds sont des éléments de $\text{dom}(D_i)$ et la racine est ALL_i^1 . Plus précisément, nous représentons par $x \in \text{dom}(D_i^p)$ le fait qu'un élément $x \in \text{dom}(D_i)$ soit une instance du niveau D_i^p et désignons par $\text{Level}(x)$ le niveau de la hiérarchie pour lequel x est une instance. Comme souvent, les arêtes d'un tel arbre H_i peuvent être vues comme des relations de type *is-a*. La relation de spécialisation (resp. la relation de généralisation) correspond à un chemin "top-down" (resp. "bottom-up") dans H_i . Un chemin connecte deux nœuds lors d'un parcours de H_i de la racine vers les feuilles (resp. des feuilles vers la racine). Lorsqu'il n'y a pas de hiérarchie définie sur une dimension D_i , nous considérons H_i comme un arbre équilibré dont la racine est ALL_i et les feuilles sont tous les éléments de $\text{dom}(D_i) \setminus ALL_i$.

Ces notations permettent de définir quelques fonctions de manipulation des éléments d'une hiérarchie. Soit $x \in \text{dom}(D_i)$ et H_i la hiérarchie associée à D_i . Nous notons x^\uparrow l'ensemble des généralisations possibles de x selon la hiérarchie spécifiée. Notons que tout élément est inclus dans sa généralisation et donc, $ALL_i^\uparrow = \{ALL_i\}$.

Exemple 2.1 Soit la dimension Lieu du cas d'étude utilisé dans ce manuscrit, nous avons : $France^\uparrow = \{France, Europe, ALL\}$.

Pour une dimension D_i et un niveau de granularité D_i^p donné, nous proposons la fonction $(D_i^p)^\#$ qui, pour $x \in \text{dom}(D_i)$, retourne la généralisation de x au niveau D_i^p . Plus formellement,

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

Exemple 2.2 Sur la même dimension Lieu, nous avons :

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

Remarque 2.1 L'extension de ces fonctions au contexte multidimensionnel étant triviale, nous ne les définissons pas ici formellement mais sont supposées connues durant la suite de ce manuscrit.

Soit $\mathcal{M} = \{M_1, M_2, \dots, M_k\}$ un ensemble de mesures où chaque mesure M_i est définie sur un domaine potentiellement infini de valeurs, noté $\text{dom}(M_i)$. Une cellule c est $(n+k)$ -uplet $etestnotec = (d_1, \dots, d_n, m_1, \dots, m_k)$ avec $d_i \in \text{dom}(D_i)$ ($1 \leq i \leq n$) et $m_j \in M_j$ ($1 \leq j \leq k$). Intuitivement, une cellule permet de désigner les valeurs des mesures m_1, \dots, m_k lorsque les dimensions valent d_1, \dots, d_n . Un cuboïde est un ensemble de cellules dont les valeurs de chaque dimension sont

1. Nous remarquons que ALL_i désigne à la fois un élément du domaine de D_i et un niveau de précision dans la hiérarchie associée.

définies sur le même niveau de précision. Formellement, un cuboïde $C = (D_1^{l_1}, \dots, D_n^{l_n})$ est défini par l'application suivante :

$$D_1^{l_1} \times D_2^{l_2} \times \dots \times D_n^{l_n} \rightarrow M_1 \times \dots \times M_k$$

avec $0 \leq l_i \leq \max_i$ pour $1 \leq i \leq n$.

Le cuboïde dont toutes les dimensions sont définies sur le plus fin niveau de précision est appelé *cuboïde de base*. Les cuboïdes associés aux différents niveaux de hiérarchies des dimensions de \mathcal{D} composent un treillis défini de la façon suivante :

Définition 2.1 (*Treillis des cuboïdes*) Soit $G = (\mathcal{C}, \leq_G)$ un treillis où \mathcal{C} est l'ensemble des cuboïdes associées aux hiérarchies H_i et \leq_G est une relation d'ordre partiel telle que pour $C = (D_1^{l_1}, \dots, D_n^{l_n})$ et $C' = (D_1^{l'_1}, \dots, D_n^{l'_n})$, nous avons $C \leq_G C'$ si $\forall i \in [1, n]$, $D_i^{l'_i} \succ D_i^{l_i}$. De plus, $\forall C, C'$, leur borne inférieure et leur borne supérieure sont définies ainsi :

- $Inf(C, C') = (\min(D_1^{l_1}, D_1^{l'_1}), \dots, \min(D_n^{l_n}, D_n^{l'_n}))$
- $Sup(C, C') = (\max(D_1^{l_1}, D_1^{l'_1}), \dots, \max(D_n^{l_n}, D_n^{l'_n}))$

Exemple 2.3 Si l'on considère les dimensions *Lieu* et *Vecteur* de communication et leur hiérarchie présentées dans le chapitre précédent, alors la figure 2.1 représente le treillis des cuboïdes associé.

Finalement, un cube de données \mathcal{C} associé aux ensemble \mathcal{M} et \mathcal{D} peut être considéré comme l'union des cuboïdes existants sur les dimensions de \mathcal{D} .

Définition 2.2 (*Cube de données*) Un cube de données \mathcal{C} est une application de n dimensions d'analyse D_1, \dots, D_n dans un ensemble de mesures $\mathcal{M} = \{M_1, \dots, M_k\}$.

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

Pour illustrer ces concepts, considérons la figure 2.2 basée sur le cas d'étude utilisé dans ce manuscrit. Celle-ci présente un cube de données défini sur trois dimensions, dont deux sont hiérarchisées (*i.e.* les dimensions *Lieu* et *Vecteur de communication*). $\langle Vecteur, Ville, Mois \rangle$ et $\langle Catégorie vecteur, Ville \rangle$ sont deux cuboïdes. Le premier est le *cuboïde de base* et stocke le nombre de messages envoyés par vecteur de communication, par ville d'envoi et par mois alors que le second matérialise le nombre de message envoyés par catégorie de vecteurs et par ville tout mois confondus. $\langle (SMS, Paris, Décembre 2010), 26 \rangle$ est une cellule du cuboïde de base signifiant que 26 SMS ont été envoyés depuis Paris au cours du mois de décembre 2010.

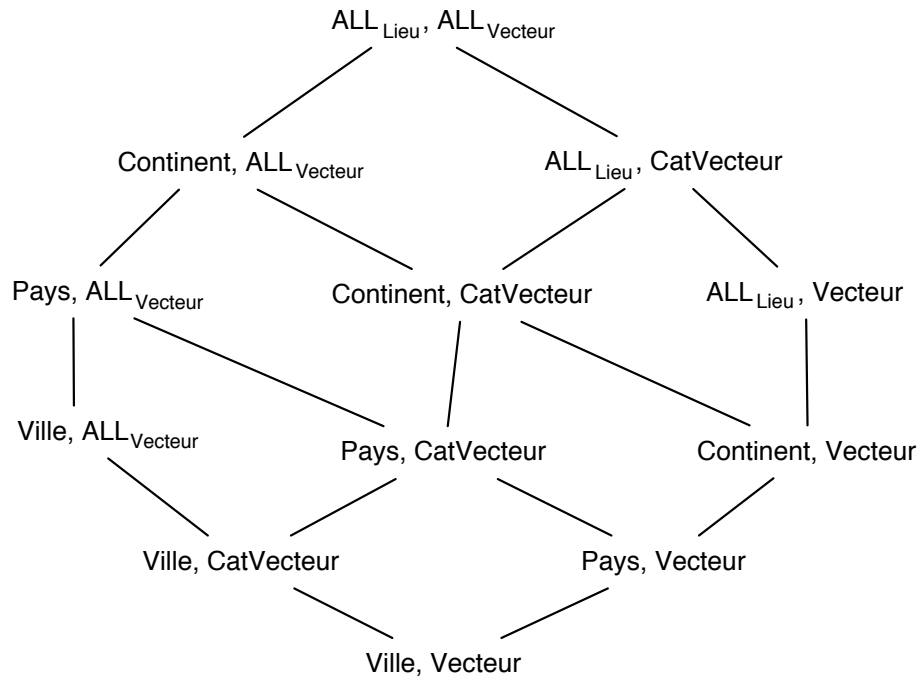


FIGURE 2.1 – Le treillis des cuboïdes associé aux dimensions *Lieu* et *Vecteur*

1.2 Bases de données multidimensionnelles

La construction efficace des cubes de données est un problème de recherche majeur dans le processus OLAP [GBLP96]. De nombreuses approches ont donc été proposées pour répondre au principal défi d'une telle construction : trouver le meilleur compromis entre la taille du cube matérialisé et le temps de réponse à une requête. En effet, matérialiser tous les cuboïdes du treillis permettrait des réponses immédiates aux requêtes mais est trop coûteux en espace de stockage. À l'inverse, matérialiser uniquement le *cuboïde de base* est avantageux d'un point de vue matérialisation mais engendrerait des temps de réponses inacceptables pour les utilisateurs. Ces deux aspects doivent pourtant être conciliés.

Nous dressons donc ici un panorama des différentes approches existantes en les classant en 3 catégories : les approches basées sur des approximations, celles dites *MOLAP* (*Multidimensional On-Line Analytical Processing*) et les approches *ROLAP* (*Relational On-Line Analytical Processing*).

Les approches basées sur des approximations partent du principe qu'il n'est pas indispensable pour un système de fournir des résultats très précis pour aider à la décision. Cette hypothèse permet donc de réduire l'espace de stockage nécessaire puisque seule une description approximative du cube est stockée. Ces approches utilisent différentes techniques telles que la transformation

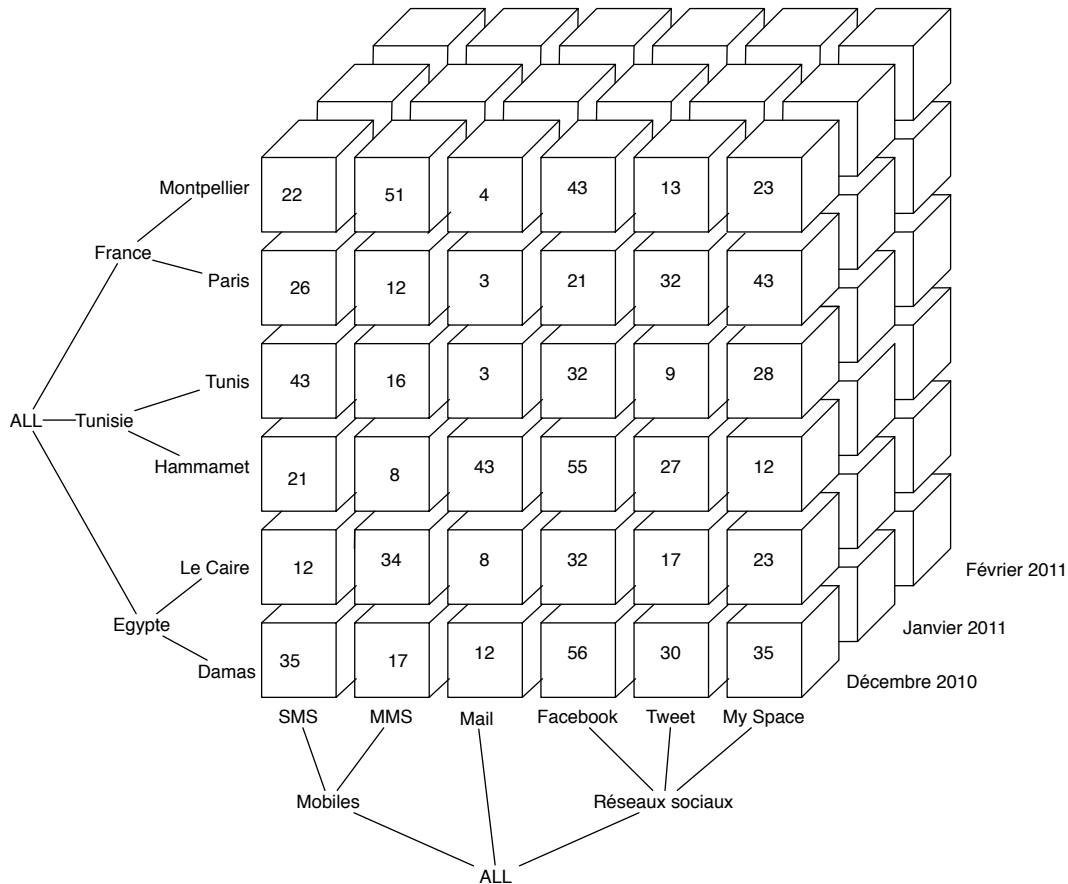


FIGURE 2.2 – Représentation graphique d'un cube de données

en ondelettes [VW99, VWI98], les polynômes multivariés [BS97] (le cube est découpé en zones qui sont approximées via un modèle statistique utilisant une régression linéaire), l'échantillonnage [AGP00], les histogrammes [PG99] ou d'autres [GKTD00].

Les approches MOLAP stockent le cube sous la forme de tableaux multidimensionnels. Ainsi, pour chaque mesure du cube, il n'est pas nécessaire de stocker les dimensions associées puisqu'elles sont déterminées directement par la position de la mesure dans le tableau. L'inconvénient majeur de ces approches est qu'en pratique les cubes sont peu denses. Par conséquent, beaucoup de cellules du tableau seront vides ce qui dégrade les performances de stockage. Le caractère épars des cubes a motivé la proposition de Multi-way [ZDN97]. Le cuboïde de base est stocké dans un tableau qui est partitionné pour réduire l'utilisation de la mémoire vive (toutes ses parties ne sont pas nécessaires en même temps). Ce découpage couplé à un plan d'exécution Top-down (le cube est construit en partant du cuboïde de base jusqu'au cuboïde le plus abstrait) permettent la réutilisation des valeurs agrégées pour calculer plusieurs cuboïdes descendants en une seule passe. Par exemple, avec 4 dimensions (A, B, C et D), le calcul de ACD permet également le calcul de AD et A. Les expérimentations menées montrent que cette approche obtient de bons résultats quand le produit des cardinalités des dimensions n'est pas élevé. Dans le cas contraire, les parties

de tableau ne tiennent pas en mémoire vive. Cette approche a inspiré MM-Cubing [SHX04] qui propose d'utiliser une technique similaire pour les sous-ensembles denses du jeu de données. En effet, les expérimentations menées sur de nombreuses propositions montrent que le paramètre ayant le plus d'impact sur les performances est la densité du jeu de données. MM-Cubing s'appuie sur cette observation et applique différentes techniques sur le jeu de donnée en fonction de la densité de la région étudiée.

Nous nous intéressons maintenant aux approches ROLAP (*Relational OLAP*) qui stockent et modélisent les bases de données multidimensionnelles à l'aide d'un modèle relationnel. Cette technique a l'avantage de ne pas nécessiter d'investissement financier supplémentaires puisque ce modèle tire parti des ressources logicielles et matérielles existantes. Nous présentons quelques approches existantes.

- BUC (*Bottom Up Computation of Sparse and Iceberg CUBE*) [BR99] construit le cube en débutant par le cuboïde le plus général. Le principe de la construction est le suivant : BUC débute par la première dimension et effectue un partitionnement selon les valeurs de cette dimension. Pour chaque élément de la partition, BUC calcule récursivement les dimensions restantes. Cette approche *Bottom-Up* (le cube est construit du cuboïde de base vers le cuboïde le plus général) permet un élagage similaire à la *Apriori* [AS94] pour le calcul d'*iceberg cubes*. Un *iceberg cube* est un cube où les cellules sont calculées si et seulement si elles respectent la *condition de l'iceberg*. Un exemple classique de cette condition est qu'une cellule est matérialisée si elle apparaît au moins x fois dans la base où x est un seuil fixé a priori. Cette condition est proche du support minimum (*minSupp*) dans les règles d'association (e.g. une règle est considérée comme fréquente si elle apparaît au moins *minSupp* fois dans le jeu de données). Une conséquence de la propriété d'anti-monotonie [AS94] (si une cellule c ne satisfait pas la condition de l'iceberg, alors aucune des cellules plus spécifique que c ne la satisfaira) permet d'élaguer efficacement l'espace de recherche. Cette approche obtient de bons résultats sur des jeux de données peu denses mais les performances se dégradent considérablement quand le jeu de données est plus complexe (comportant des zones denses et d'autres éparses). Nous notons également que cette approche ne prend pas en compte les dimensions hiérarchisées.
- CURE [MI06] est une proposition s'inspirant de BUC puisque le cube est également construit selon un plan d'exécution *Bottom-Up*. Contrairement à BUC, (1) CURE ne propose pas la construction d'*iceberg cube* et (2) grâce à un plan d'exécution judicieux, cette approche prend en compte les dimensions hiérarchisées. En effet, dans la plupart des applications réelles, les dimensions sont observables à plusieurs niveaux de précision. De plus, les décideurs ont be-

soin de cette vision multiniveaux pour dégager des tendances, détecter des exceptions, . . . La prise en compte des hiérarchies est donc un atout important. Notons que, tout comme dans [ZDN97], un partitionnement du cube permet de ne stocker en mémoire vive que les parties du cube nécessaires à un moment donné.

- H-Cubing [HPDW01] utilise une structure de hachage arborescente (*H-tree*) pour faciliter la construction et le stockage du cube. Dans cette approche, les dimensions hiérarchisées ne sont pas prises en compte. Cette structure autorise les méthodes *Top-Down* et *Bottom-Up* pour calculer le cube et réduit par nature la redondance des données. Toutefois, comme [BR99], il n'est pas possible d'utiliser les résultats intermédiaires pour le calcul simultané des cuboïdes.
- Pour accélérer la construction du cube, StarCubing [XHLW03] intègre dans une seule proposition les forces des approches *Top-down* (calcul simultané de plusieurs cuboïdes) et *Bottom-up* (élagage à la *Apriori*). Cette approche s'appuie sur une extension de la structure de hachage arborescente proposée dans [HPDW01] appelée Star-Tree. Cette structure permet une compression sans perte du cube grâce à la suppression des nœuds ne satisfaisant pas la *condition de l'iceberg*.
- [PMC05] constate que toutes les approches proposées pour construire le cube s'appuient sur une seule table regroupant tous les n-uplets. En pratique, cette situation est rare puisqu'un entrepôt de données est souvent découpé en plusieurs tables. Par conséquent, les approches existantes nécessitent le calcul d'une jointure conséquente qui dégrade les performances. [PMC05] propose alors un algorithme efficace, CTC (*Cross Table Cubing*), pour construire des *iceberg cubes* à partir de plusieurs tables.

Dans cette section, nous nous sommes intéressés aux méthodes de construction d'un cube de données dans un environnement statique. L'inadéquation de ces techniques dans un contexte de flot est discuté dans la section 4. Malgré tout, nous verrons au cours de la section 4, qu'il existe quelques approches pour construire un cube alimenté par un flot. La prochaine section est consacrée à l'autre pendant de la problématique de ce travail : les flots de données.

2 Manipuler des flots de données

2.1 Modèles de flots

Traditionnellement, il est reconnu qu'il existe trois principaux modèles de flots [Agg07]. Bien entendu, le modèle utilisé dépend des besoins de l'utilisateur ainsi que de l'application finale. Pour décrire ces trois modèles, nous utilisons les notations et définitions de [Mut05]. Ces notations et

définitions seront ensuite modifiées pour intégrer l'aspect multidimensionnel des données circulant dans les flots considérés dans ce manuscrit. Notons F un flot de données. Les éléments présents dans ce flot sont notés a_1, a_2, \dots (où a_1 est l'élément le plus ancien du flot) et décrivent un signal sous-jacent à p variables $A : [0, \dots, p - 1]$ à valeurs dans \mathbb{R} . Les trois modèles diffèrent uniquement dans la manière dont les éléments a_i décrivent le signal A :

- Le modèle *des séries temporelles* : dans ce modèle, chaque élément $a_i = A[i - 1]$. Il existe donc une adéquation parfaite entre le flot de données et le signal qu'il décrit.
- Le modèle *de la caisse enregistreuse* : dans ce modèle, chaque élément a_i vient s'ajouter à $A[j]$. En fait, on peut définir chaque élément $a_i = (j, I_i)$ avec I_i positif et $A_i[j] = A_{i-1}[j] + I_i$. D'un point de vue sémantique, ce modèle tient compte de l'augmentation du signal pour une valeur $j \in [0, \dots, p - 1]$ à chaque instant i . Ce modèle est le plus utilisé dans les applications de flots de données.
- Le modèle *du tourniquet* : dans ce modèle, et contrairement au modèle précédent de la caisse enregistreuse, chaque élément peut soit s'ajouter soit se soustraire à la valeur précédente dans $A[j]$. Ainsi, pour chaque élément $a_i = (j, I_i)$ avec I_i positif ou négatif, $A_i[j] = A_{i-1}[j] + I_i$. Ce modèle est le plus général possible mais est en pratique peu utilisé par les applications en raison notamment du calcul des bornes sur les variations du signal A .

2.2 Gestion du temps dans les flots

Le temps est une caractéristique inhérente au concept de flot de données. En effet, dans la plupart des flots de données, chaque élément arrivant sur le flot possède une estampille temporelle. Le cas échéant, il est toujours possible de générer cette estampille s'appuyant sur la date de réception de l'élément par le système traitant ce flot. Dans la mesure où la taille d'un flot de données est non bornée et potentiellement infinie, il n'est pas réaliste, comme nous l'avons vu dans l'introduction de ce manuscrit, de stocker et traiter l'intégralité d'un flot. Il est donc essentiel de restreindre le traitement appliqué sur un flot à une portion de celui-ci. Dans la littérature, cette portion est généralement appelée *fenêtre* [Agg07, Mut05, GZK05]. Nous décrivons ci-dessous les principaux types de fenêtres existants :

- *La fenêtre fixe* : cette fenêtre représente une portion figée du flot de données. Une fenêtre fixe possible concernant le cas d'étude utilisé dans ce manuscrit pourrait être *les ventes du mois de Décembre*.
- *La fenêtre glissante* : cette fenêtre est définie par deux bornes dynamiques mises à jour selon deux conditions possibles. Soit ces bornes sont réévaluées après chaque nouvelle arrivée d'un élément (réévaluation avide), soit ces bornes sont réévaluées après une période de temps définie par l'utilisateur. Il existe une variante des fenêtres glissantes : les fenêtres sautantes

(appelées aussi *batches* dans la littérature). Un découpage du flot en fenêtres sautantes désigne un découpage tel que chaque élément du flot appartient à une et une seule fenêtre. Formellement, un flot F peut donc être défini comme une séquence potentiellement infinie de batches, $F = B_0, B_1, \dots$ où chaque batch est un ensemble d'éléments consécutifs du flot. Dans un flot, la taille de chaque batch (*i.e.* le nombre d'éléments les composant) peut être variable en fonction du mode de découpage choisi. Soit le découpage est effectué à intervalle de temps fixe auquel cas les batches auront une taille différente si le débit du flot est variable. Soit le flot est découpé selon un nombre d'éléments fixe même si le débit du flot varie.

- 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*) [GHP⁺02] est utilisé pour modéliser et compresser la dimension temporelle. 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 vieillissent, plus le niveau de précision diminue. La vitesse à laquelle la précision diminue dépend du domaine d'application. La figure 2.3 expose trois exemples de *tilted-time window*. En effet, [GHP⁺02] propose deux modèles différents. Le premier (2.3(a) et 2.3(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 2.3(a) stockera pendant une semaine les ventes quotidiennes. Ensuite, pendant un mois, les ventes seront stockées hebdomadairement 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 2.3(a) stockera seulement 18 valeurs (7+4+3+4). Le second modèle de *tilted-time* (2.3(c)) effectue un découpage logarithmique. Dans le cas d'un logarithme en base 2 (comme sur la figure), pour représenter 365 valeurs il faudrait seulement $\log_2(365)$ valeurs. Les *tilted time windows* sont mises à jour par effet de cascade comme illustré dans la figure 2.4. Lorsque le niveau de granularité le plus bas est rempli (ici ce niveau peut contenir trois éléments au maximum), ces éléments sont regroupés en utilisant une fonction d'agrégation définie par l'utilisateur (dans notre exemple nous avons arbitrairement choisi la somme). Le résultat de cette agrégation est alors inséré au niveau de granularité suivant et le premier niveau est ainsi vide. Si le second niveau de granularité est plein, un processus similaire d'agrégation-décalage se produit et ainsi de suite jusqu'au dernier niveau. Tout au long de ce manuscrit, les *tilted time window* seront largement utilisées, adaptées et discutées. Nous présentons maintenant les notations et la terminologie que nous utiliserons dans la suite de ce document. Formellement, une *tilted time window* $T = \langle W_1, W_2, \dots, W_n \rangle$ est définie comme une suite de fenêtre W_i . Chaque fenêtre représente un niveau de granularité et W_1 est la fenêtre de granularité la plus fine (*i.e.* celle recevant les valeurs brutes du flot). On note $|T|$ le nombre

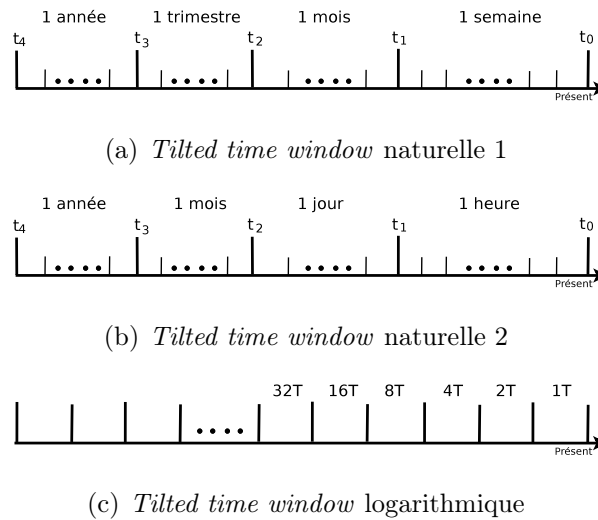


FIGURE 2.3 – Exemples de *tilted time windows*

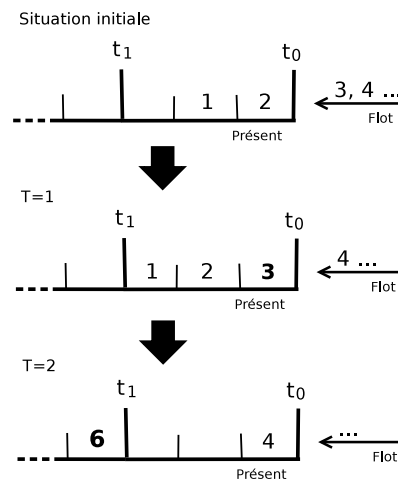


FIGURE 2.4 – Illustration du mécanisme de mise à jour d'une *tilted time window*

de fenêtre de T et $card(W_i)$ désigne le nombre d'éléments que peut accueillir la fenêtre W_i . Dans la mesure où un élément de F peut être le résultat d'une agrégation d'éléments "bruts", nous introduisons la fonction $Repr(W_i)$ pour quantifier le nombre d'éléments du flot représentés par une valeur stockée dans une fenêtre W_i de T . Pour une fenêtre W_i donnée, $Repr(W_i)$ peut être calculé grâce au schéma de récursion suivant.

$$Repr(W_i) = \begin{cases} 1 & \text{si } i = 1 \\ Repr(W_{i-1}) \times Card(W_{i-1}) & \text{sinon} \end{cases}$$

Par exemple, si l'on considère la *tilted time window* présentée dans la figure 2.3(a) et en supposant $\text{Card}(W_1) = 7$, nous avons $T = \langle W_1, W_2, W_3, W_4 \rangle$ avec $|T| = 4$ et $\text{Repr}(W_2) = 1 \times 7 = 7$ (*i.e.* chaque élément stocké dans W_2 représente 7 éléments du flot de données).

Nous définissons la fonction $Fenetre(t, t_{deb}, t_{fin})$ qui permet d'indiquer quelles sont les fenêtres à considérer si l'on veut connaître l'historique des données entre t_{deb} et t_{fin} à la date t .

Dans cette section, nous avons présenté les principaux modèles de flots existants ainsi que les différentes stratégies pour gérer la temporalité des données en fonction des besoins. Nous consacrons la prochaine section à dresser un panorama des méthodes existantes pour résumer un flot de données.

3 Panorama des techniques de résumé de flots de données

Les approches de résumé de flots peuvent généralement être regroupées en trois catégories :

1. Les méthodes statistiques pour résumer des séries temporelles,
2. Les techniques de fouille de données,
3. Les techniques de construction de cubes de données.

Dans la suite de cette section nous dressons un panorama des approches relatives à ces trois catégories. Néanmoins, la première catégorie étant assez éloignée de notre problématique, nous ne nous y intéressons que brièvement.

3.1 Utilisation de méthodes statistiques

Un flot de données numériques peut être considéré comme une série temporelle dont la taille croît constamment. Dès lors, il fut naturel d'appliquer les méthodes issues de la théorie du signal pour compresser de telles données. Parmi les outils mathématiques efficaces et robustes utilisés, nous pouvons citer les ondelettes [CF02, PM02], les transformations de Fourier discrètes [FRM94, RM97], les transformées en cosinus discrètes, . . . Bien que ces techniques soient très adaptées dans un contexte de résumé de flots de données numériques, celles-ci s'éloignent de notre problématique pour la raison suivante. Même s'il existe quelques approches pour traiter des données multidimensionnelles [MS04, TGIK02], les données manipulées par la plupart de ces approches ne comportent généralement qu'une seule dimension numérique et non hiérarchisée. Par conséquent, nous ne discuterons pas plus ces approches dans le reste de ce chapitre.

3.2 Utilisation de techniques de fouille de données

D'une certaine manière, les approches de fouille de données sont assimilables à des techniques de résumé dans le sens où la connaissance extraite permet de capturer les tendances dans le flot. Nous présentons maintenant quelques adaptations de techniques de fouille au contexte dynamique des flots de données.

Clustering (ou segmentation) Dans un environnement statique, le clustering d'éléments consiste à déterminer, sans connaissances préalables, des groupes d'éléments possédant des caractéristiques communes. Cependant, les approches de clustering classiques sont inadaptées car elles exigent plusieurs passages sur la base de données. Nous présentons ici deux approches adaptées au contexte des flots.

Jusqu'à CLUSTREAM [AHWY03], les approches existantes ne prenaient pas en compte l'évolution des données. Par conséquent, les clusters construits devenaient inadaptés quand la distribution des données évoluait de façon importante au cours du temps. CLUSTREAM résout ce problème et autorise l'exploration des clusters sur différentes parties du flot. Cet algorithme fonctionne en deux étapes : la première dite *en ligne* stocke périodiquement des statistiques de résumés détaillés (micro-classes). Tout nouvel élément du flot est intégré à la micro-classe la plus proche. Si aucune n'est suffisamment proche de cet élément, une nouvelle micro-classe est créée et deux autres sont fusionnées garantissant une utilisation de la mémoire fixe. La seconde étape, dite *hors ligne*, utilise les informations statistiques à des fins d'analyse.

BIRCH [ZRL06] (Balanced Iterative Reducing and Clustering using Hierarchies) est un algorithme adapté à des données volumineuses. L'approche est basée sur une classification effectuée sur un résumé compact des données au lieu des données originales. BIRCH peut ainsi traiter un grand volume de données en utilisant une mémoire limitée. De plus, il s'agit d'une approche incrémentale qui nécessite une seule passe sur les données.

Classification supervisée Dans un environnement statique, la classification permet d'affecter des éléments à une classe prédéterminée. Une phase d'apprentissage peut être nécessaire pour déterminer les caractéristiques de chaque classe. Par exemple, la classification de textes en *article scientifique* ou *recette de cuisine* demande l'apprentissage des modèles de chaque catégorie grâce à un ensemble de textes déjà étiquetés. Cet ensemble de textes déjà étiqueté est appelé *jeu d'apprentissage*.

La plupart des propositions de classification sur les flots de données ne répondent pas simultanément aux trois principales contraintes induites par les flots (vitesse, mémoire limitée et évolution de la distribution des données). Nous choisissons de présenter uniquement On-Demand Classification [AHWY04] qui réalise un tel compromis.

On-Demand Classification [AHWY04] reprend l'idée de micro-classes introduites dans CLUSTREAM. Cette approche se décompose en deux étapes. La première stocke de façon continue des statistiques résumées sous forme de micro-classes. La seconde exploite ces statistiques pour effectuer la classification. Le terme *On-Demand* vient du fait que les deux étapes sont réalisables en ligne.

Découverte de motifs fréquents La problématique de l'extraction de motifs fréquents date d'une vingtaine d'années environ (*i.e.* les premiers travaux proposés datent du milieu des années 80 et concernent l'extraction d'itemsets) et pendant ces années les principaux travaux se sont principalement intéressés à des bases de données statiques. Le développement des applications autour des flots de données a nécessité de revoir considérablement les approches précédentes dans la mesure où ces dernières nécessitaient d'avoir accès à l'intégralité de la base de données. L'extraction de motifs dans des flots permet d'offrir de nouveaux types de résumés qui sont particulièrement adaptés à des applications de supervision où l'on recherche par exemple quels sont les éléments qui apparaissent le plus dans le flot. Même s'il existe de très nombreux motifs (*e.g.* items, itemsets, séquences, épisodes, arbres, graphes) que l'on peut extraire, l'objectif de cette partie est de se focaliser sur deux principaux : les itemsets (utilisées notamment dans le cas des règles d'association) et les motifs séquentiels.

Li et al. [LLS04] utilisent une extension d'une représentation basée sur un ensemble d'arbres (*Item-Suffix Frequent Itemset Forest*) dans leur algorithme DSM-FI. Cet algorithme propose aussi une approximation de l'ensemble des itemsets fréquents. Ainsi, les auteurs proposent de relaxer le seuil de fréquence minimal permettant ainsi de mieux garantir la précision de l'extraction. Tous les sous-itemsets extraits sont stockés dans une forêt d'arbres suffixés et de tables d'en-têtes. Les auteurs soulignent que cette représentation par forêt est plus compacte que la représentation par arbre préfixé, malheureusement ce gain en espace mémoire a comme conséquence un plus grand temps de calcul.

Les auteurs de [CL03], quant à eux, proposent de ne travailler que sur la partie récente du flot et introduisent ainsi une approche basée sur le taux d'oubli afin de limiter et de diminuer au fur et à mesure du temps le poids des anciennes transactions. Ainsi l'algorithme permet de se concentrer uniquement sur les parties récentes du flot, mais ce principe d'oubli introduit un trop grand taux d'erreur sur la fréquence et peut s'avérer difficile à gérer dans le cas d'applications critiques.

La recherche de séquences fréquentes (*sequential patterns* en anglais) sur des bases de données statiques a donné lieu à un grand nombre de propositions. Cette recherche vise à faire émerger des comportements fréquents en recherchant des corrélations entre attributs et en prenant en compte

le temps. Par exemple, $\langle (raquette, tennis)(balles) \rangle$ est une séquence fréquente si plus de $x\%$ (où x est un seuil fixé a priori) des clients achètent une raquette et des tennis puis reviennent au magasin acheter des balles. *raquette* est un *item* et $(raquette, tennis)$ un *itemset*. Peu d'algorithmes ont été proposés pour l'extraction de séquences fréquentes sur les flots de données. Nous présentons brièvement ci-dessous quelques approches pour les extraire :

Dans [MM06], Marascu et al. proposent l'algorithme SMDS (Sequence Mining in Data Streams). L'algorithme SMDS repose sur une approche par segmentation (clustering) de séquences. Des ensembles de séquences similaires sont alors construits au fur et à mesure de l'avancée du flot.

[RPT06] proposent un algorithme d'extraction de motifs séquentiels maximaux sur les flots de données appelé SPEED. L'idée principale de l'approche est de conserver les séquences potentiellement fréquentes (séquences sous-fréquentes). Grâce à une représentation des itemsets au moyen d'un entier et à l'association d'un nombre premier pour chaque item, il est possible d'utiliser des propriétés d'arithmétiques pour les tests d'inclusions de séquences.

Une troisième approche d'extraction de motifs séquentiels est quelque peu différente des deux précédentes puisqu'elle prend en compte l'extraction sur un ensemble de flots de données. Les auteurs Chen et al. [CWZ05] considèrent ainsi que les motifs extraits de cet ensemble de flots de données sont des motifs multidimensionnels. Les auteurs choisissent un modèle de fenêtre glissante pour observer le flot. Leur algorithme, nommé MILE, utilise l'algorithme PrefixSpan [PHMa⁺01] à chaque étape de glissement afin d'extraire ces motifs.

A notre connaissance, seul [RP08] permet l'extraction de motifs séquentiels multidimensionnels fréquents dans l'ensemble du flot de données. Le découpage du temps choisi est le batch. L'approche fonctionne en deux temps. Dans un premier temps, l'ensemble des items multidimensionnels les plus spécifiques est extrait afin de limiter l'espace de recherche. Ensuite, les motifs séquentiels multidimensionnels sont extraits en utilisant l'algorithme PrefixSpan [PHM⁺04] et en n'exploitant uniquement les items les plus spécifiques précédemment extraits. Il est important de noter que bien que cette approche permette l'extraction de motifs séquentiels multidimensionnels, nous ne considérons pas que ces motifs sont multiniveaux. En effet, pour chaque dimension D_i considérée, les seules valeurs possibles pouvant apparaître dans un motif extrait sont soit des items apparaissant dans le flot (*i.e.* des items de plus faible granularité) soit une valeur joker (notée \star) équivalente à ALL_i .

3.3 Utilisation de techniques de construction de cubes de données

Finalement, la problématique de construction d'un cube de données alimenté par un flot de données multidimensionnelles est assez récente et très peu d'approches la considèrent. Nous présentons ci-dessous les principales contributions autour de cette thématique.

DAWA [HCY05]

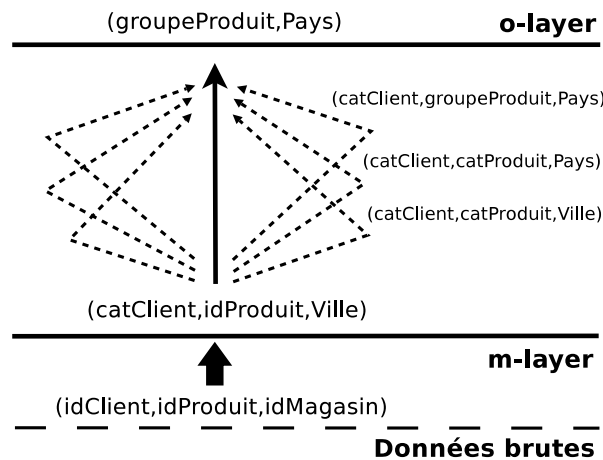
Dans [HCY05], les auteurs proposent une méthode originale pour construire un cube de données par batch du flot. Pour cela, ils combinent deux techniques d'approximations de données très utilisées dans le cadre de la construction de cube sur des données statiques : les transformées en cosinus discrètes (DCT) et les transformées en ondelettes discrètes (DWT) . En effet, la combinaison de ces deux techniques permet de remédier au problème de stockage inhérent aux DWT et au problème de temps de calcul des DCT. Malgré des propriétés intéressantes, cette approche ne considère pas que les valeurs d'une dimension puissent être observées à plusieurs niveaux de granularité. Pour cette raison, nous ne discutons pas davantage cette approche dans la suite de ce chapitre

StreamCube [HCD⁺05]

Les auteurs de [HCD⁺05] proposent une solution pour construire et mettre à jour un cube alimenté par un flot et permettre ainsi une analyse multidimensionnelle et multiniveaux. Nous la présentons donc en détail dans cette section.

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*) [GHP⁺02] est utilisé pour modéliser et compresser la dimension temporelle. Ainsi, une *tilted time window* est stockée dans chaque cellule matérialisée (*i.e.* physiquement stockée).

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, [HCD⁺05] 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*). 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

FIGURE 2.5 – Les différents niveaux de précision de [HCD⁺05]

consulté permettant au décideur de détecter les tendances globales et les exceptions. De manière à pouvoir répondre aux 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. [HCD⁺05] 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 $(idClient, idProduit, idMagasin)$ (figure 2.5).

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*.

Maintenant que nous avons présenté les principales approches de résumé de flots existantes, nous spécifions le cadre théorique dans lequel se situe notre travail.

4 Données manipulées

Adaptation des modèles existants au contexte multidimensionnel

Dans le cadre des travaux réalisés au cours de cette thèse, les flots de données que nous avons manipulés diffèrent légèrement de la définition classique des flots de données présentée ci-dessus. Pour intégrer la contrainte de multidimensionnalité des données, nous définissons un flot de la

manière suivante. Notons F un flot de données n -dimensionnelles. Les éléments présents dans ce flots sont notés $a_1, a_2, | \dots$ (où a_1 est l'élément le plus ancien du flot). Chaque élément a_i est décrit selon un vecteur à n dimensions. Par exemple $a_i[j]$ (avec $j \in [0, \dots, n - 1]$) désigne la valeur associée à la $j + 1^{\text{ème}}$ dimension de l'élément a_i du flot F . Le domaine de définition associé à chacune de ces dimensions n'est pas nécessairement additif. Par exemple, en considérant le cas d'étude présenté au chapitre 1, une dimension possible est la dimension *Lieu*. Le domaine de définition de cette définition étant catégoriel, une somme entre deux membres de cette dimension n'est pas définie. Les éléments de F décrivent un signal sous-jacent à p variables $A : [0, \dots, p - 1]$ tels que chaque élément $A[i]$ est aussi un vecteur à n dimensions. En s'appuyant principalement sur la possible non-additivité des membres de chaque dimension, nous pouvons immédiatement exclure l'utilisation des modèles de la caisse enregistreuse et du tourniquet dans le cadre de cette thèse. Ainsi, tout au long de ce manuscrit, le modèle de flots de données utilisé sera le modèle des séries temporelles².

Données circulant dans le flot

Tout au long de ce manuscrit, nous considérons que les données circulant dans le flot sont de la forme suivante. Soit $e = (d_1, \dots, d_n, m)$ un *item multidimensionnel* du flot qui correspond à une cellule du cuboïde de base $(D_1^{max_1}, \dots, D_n^{max_n})$ et m est la mesure associée à cette cellule. Nous remarquons ici qu'une seule mesure est considérée dans ce travail. Dans la plupart des approches proposées dans ce manuscrit, l'extension des définitions et approches à un contexte multi-mesure est immédiate. Le cas échéant, nous discutons comment prendre en compte plusieurs mesures dans les chapitres associés. De plus, nous supposons que pour chaque dimension D_i , la hiérarchie associée est un arbre ou bien un graphe orienté sans cycle où le chemin d'agrégation est supposé fixé.

Gestion du temps

Tout au long de ce manuscrit, le mode de découpage du flot adopté est le *batch*. Nous considérons ainsi un flot $F = B_0, B_1, \dots$ comme une suite non bornée et potentiellement infinie de batches tel qu'un batch est une suite d'items multidimensionnels.

5 Discussion

Maintenant que nous avons présenté un panorama des travaux connexes à notre problématique ainsi que le type de données que nous considérons dans ce travail, nous discutons ces travaux.

2. La définition de ce modèle reste inchangée dans notre contexte.

Pour cela, nous définissons, dans un premier temps, quelques critères d'évaluation sous-jacents aux différents domaines abordés dans cette thèse. Ensuite, nous discutons des travaux décrits précédemment sous cet éclairage. Enfin, nous réalisons une synthèse de cette discussion faisant émerger les défis à relever.

5.1 Critères d'évaluation

Dans cette section, nous discutons de l'adéquation à notre problématique des techniques présentées dans la section précédente en nous basant sur les critères suivants que nous exposons et justifions :

1. Critères relatifs à l'aspect entrepôt de données
 - Prise en compte de données multidimensionnelles (et symboliques) : Dans la mesure où les données que nous manipulons sont multidimensionnelles par nature, nous nous attachons à considérer ce critère ;
 - Prise en compte des hiérarchies : Une des préoccupations majeures au cours de cette thèse a été de proposer des méthodes de résumé exploitant efficacement les hiérarchies, ce critère apparaît essentiel.
2. Critères relatifs à l'aspect flot de données
 - Traitement de données *à la volée* : Ce travail est à la frontière de divers thèmes de recherche. Malgré tout, les données que nous manipulons dans ce travail sont disponibles sous la forme d'un flot de données. Dès lors, nous discutons des approches présentées dans la section précédente sous l'éclairage de la contrainte du traitement *à la volée* des données ;
3. Critères d'évaluation d'un résumé de flot de données
 - Compacité : Par opposition à un flot de données, la taille d'un résumé doit être bornée. Outre les considérations de stockage, ce critère joue un rôle crucial dans la phase d'analyse du résumé. En effet, à mesure que le résumé grandit, un utilisateur désirant l'exploiter doit déployer de plus en plus d'efforts pour l'appréhender correctement et prendre les décisions adéquates ;
 - Mise à jour rapide : Afin de ne pas "bloquer" le flot, le temps de traitement d'une donnée ou d'un ensemble de données doit donc être plus rapide que le débit du flot
 - Représentativité : Une des caractéristiques essentielles que doit posséder un résumé est sa représentativité par rapport au flot réel de données. Nous verrons dans la suite de cette section et dans les chapitres suivants que ce critère n'est pas toujours aisé à définir et à évaluer.

5.2 Discussion des approches présentées

Inadéquation des méthodes de construction d'un cube de données dans un contexte de flot

Nous constatons que de nombreuses approches existent pour construire des cubes de données satisfaisant le compromis espace de stockage/temps de réponse aux requêtes. Les techniques par approximation utilisent des outils mathématiques pour produire une description fidèle et compacte des données réelles mais sortent du cadre de ce travail. Ensuite, les approches *Bottom-Up* et *Top-down* permettent d'optimiser le temps de construction et l'espace occupé par le cube. L'approche *Top-down* facilite le calcul de plusieurs cuboïdes en une seule passe alors que *Bottom-up* autorise un élagage a la *Apriori* lors de la construction d'*iceberg cubes*. Nous analysons maintenant l'adaptation possible de ces travaux dans un contexte dynamique.

Premièrement, les approches citées ci-dessus pour construire un cube de données évaluent leurs performances sur des jeux de données tenant en mémoire vive. Ce premier point est une limite majeure puisqu'il entre en forte opposition avec le caractère infini d'un flot.

Ensuite, dans un flot, les données sont observables à un faible niveau de granularité. La prise en compte des hiérarchies est donc une nécessité. Cependant, parmi les approches proposées, seule CURE [MI06] propose de répondre à ce problème.

De plus, la circulation des données du flot à un faible niveau d'abstraction motive l'utilisation d'une approche *Bottom-up* pour construire le cube. En effet, une stratégie *Top-down* impliquerait, pour chaque valeur du flot, (1) une généralisation pour se positionner sur le cuboïde le plus général et (2) une succession de spécialisations pour atteindre le cuboïde de base. Le débit élevé d'un flot rend cette approche irréalisable.

Notons qu'une stratégie *Bottom-up* n'autorise pas un élagage à la *Apriori* dans les flots de données. Pour s'en convaincre, imaginons qu'à un instant t une cellule c de mesure m ne satisfasse pas la condition de l'*iceberg*. Par définition, celle-ci n'est donc pas matérialisée. Plus tard, c peut à nouveau apparaître dans le flot avec une mesure m' . Comme m et m' sont nécessaires pour vérifier si la *condition de l'iceberg* est respectée et que m a été précédemment écarté, le calcul correct de la nouvelle mesure est impossible. Le calcul d'*iceberg cube* ne peut donc pas être envisagé en l'état.

Une approche *Bottom-up* stockant tous les cuboïdes comme dans [MI06] n'est pas envisageable car le cube construit serait trop volumineux.

Enfin, une approche MOLAP est inapplicable dans notre situation. En effet, la cardinalité des dimensions n'est pas connue a priori dans un flot. Par conséquent, construire une table multidimensionnelle comme dans [ZDN97] peut s'avérer délicat et entraîner une explosion mémoire. La densité du jeu de données n'est pas non plus connue à l'avance. Une approche telle que [SHX04]

qui découpe le jeu de données selon la densité et leur applique des techniques différentes est donc irréalisable.

Malgré de nombreuses propositions pour construire efficacement un cube de données, la transposition de ces approches à un cadre dynamique n'est pas triviale. Plusieurs contraintes doivent être respectées. Premièrement, la matérialisation totale du cube n'est pas envisageable. La construction d'un iceberg cube n'étant pas possible, il faut trouver une alternative réduisant le coût de matérialisation sans utiliser des techniques mathématiques d'approximation. Ensuite, la construction du cube doit suivre une approche *Bottom-up*. La prise en compte des hiérarchies est également une nécessité pour permettre une analyse multiniveaux efficace. Enfin, les techniques de construction de cube de données doivent absolument être testées sur des jeux de données ne tenant pas en mémoire vive pour que l'on puisse déterminer si un passage à l'échelle est possible.

Nous nous intéressons maintenant à l'analyse des techniques de résumé de flots de données.

Limites des approches de fouilles de données pour résumer un flot de données

Parmi les approches de fouille présentées dans ce chapitre, seule [RP08] permet de fouiller un flot de données multidimensionnelles mais, à notre connaissance, aucune méthode n'existe pour extraire une connaissance multidimensionnelle et multiniveaux. Par conséquent, aucune des approches présentées ici ne satisfait pleinement les critères liés à l'aspect *entrepôt de données*.

Par contre, par nature, toutes les méthodes proposées ici satisfont les critères liés à l'aspect flot de données.

Les approches de fouille de flots de données décrites dans ce chapitre peuvent être regroupées en deux catégories. La première catégorie concerne les techniques qui fouillent l'intégralité du flot de données et fournissent ainsi une connaissance générale sur l'historique du flot de données. Le principal inconvénient de ce type d'approches dans un contexte de résumé de flot de données est qu'une telle connaissance ne peut refléter les changements de distributions des données dans le flot à travers le temps. Dès lors, un tel type de résumé est inexploitable si l'utilisateur désire connaître l'évolution des données circulant dans le flot. En ce sens, le critère de représentativité du résumé n'est satisfait que très partiellement. La seconde catégorie, quant à elle, concerne les méthodes qui fouillent le flot de données sur une succession de fenêtres du flot. Ici, l'analyse des connaissances extraites successivement permet de considérer l'évolution des données dans le flot et satisfait donc plus justement le critère de représentativité du résumé. Par contre, au regard du caractère potentiellement infini d'un flot, l'ensemble des connaissances extraites pour chaque fenêtre est en constante expansion. Ainsi, ce type de résumé ne satisfait pas le critère de compacité.

Limite des méthodes de résumé de flot de données multidimensionnelles existantes

Nous discutons StreamCube [HCD⁺05] qui est la seule approche parfaitement applicable à notre contexte, elle s'inscrit naturellement comme la référence de notre travail. Même si StreamCube ouvre donc de nouvelles pistes de recherche, cette proposition nous apparaît insuffisante pour trois raisons :

- 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, [HCD⁺05] 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.
- L'utilisation du modèle des *tilted time windows* peut engendrer une perte de précision importante lorsqu'une cellule n'apparaît que très rarement dans le flot de données. En effet, le changement de granularité à intervalle fixe peut dégrader sérieusement le résumé stocké. Pour s'en convaincre, considérons la figure 2.6(a). Les points représentent les valeurs du flot et les segments représentent les valeurs contenues dans la *tilted time window* (plus la valeur stockée est éloignée par rapport au début de la *tilted time window*, plus l'intervalle de temps représenté par cette valeur est grand). Dans le cas où un élément apparaît rarement dans le flot, les agrégations stockées dans la *tilted time window* associée ne permettent pas de faire apparaître cette disparité. Pourtant, la connaissance précise de ces apparitions peut être stratégiquement intéressante pour un utilisateur. Une argumentation similaire peut expliquer la faiblesse des *tilted time windows* pour résumer des données apparaissant périodiquement dans le flot (figure 2.6(b)). Ainsi, le mécanisme des *tilted time windows* est particulièrement adapté lorsque la distribution des données est équilibrée mais présente un inconvénient majeur lorsqu'il s'agit de résumer des données dont la fréquence d'apparition est changeante.

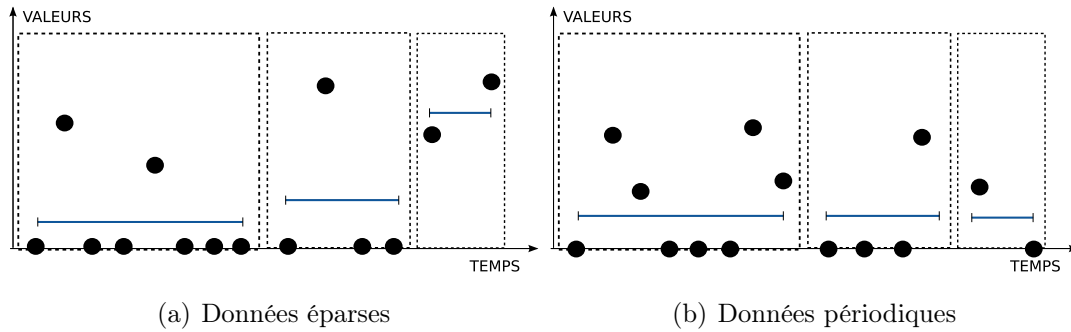


FIGURE 2.6 – Illustration de la faiblesse des TTW

Inadéquation des DSMS dans un contexte d'analyse OLAP *a posteriori*

Les DSMS (*Data Stream Management Systems*) présentent des caractéristiques intéressantes dans notre contexte³. Premièrement, ces systèmes ont dû prendre en compte le caractère infini d'un flot et offrent la possibilité de formuler des requêtes continues sur le flot de données via des langages étendant souvent SQL. Malgré ce point intéressant, ces systèmes ne sont pas adaptés à notre contexte pour deux raisons. Premièrement, il n'existe pas de DSMS pour les flots multidimensionnels. Par conséquent, une analyse multidimensionnelle et considérant les données sur plusieurs niveaux de granularité est impossible. La validation des critères "*cube*" est pourtant indispensable dans notre contexte. En outre, il est nécessaire de prévoir les requêtes continues que l'on veut appliquer au flot de données car, par principe, un DSMS ne retient pas les événements d'un flot de données s'ils ne permettent pas de répondre à une requête continue. En ce sens, les DSMS ne peuvent donc pas être considérés comme une technique de résumé de flots de données permettant une analyse *a posteriori* de son historique.

Le tableau 2.1 synthétise et conclut la discussion des travaux existants. À notre connaissance, il n'existe donc aucune proposition satisfaisant pleinement l'intégralité des critères considérés dans ce chapitre.

3. Par souci de lisibilité, nous proposons en annexe un survol des différents DSMS existants.

	Critères “cubes”		Critères “flots”		Critères “résumé”		
	Multidimensionnalité	Hierarchies	Données dynamiques	Passage à l'échelle	Compacité	Rapidité	Représentativité
1. Construction de cube sur BD statique	✓	✓					
2. Méthodes de résumé							
2.1 Statistiques			✓	✓	✓	✓	✓
2.2 Fouille de motifs							
2.2.1 Intégralité			✓	✓	✓	✓	
2.2.2 Fenêtres successives			✓	✓		✓	✓
2.3 Cube sur flot							
2.3.1 Stream Cube [HCD ⁺ 05]	✓	✓	✓			✓	
2.3.2 DAWA [HCY05]	✓		✓		✓	✓	
3. DSMS			✓	✓			

TABLE 2.1 – Synthèse de la discussion des approches connexes à la problématique de cette thèse

Chapitre 3

Window Cube, une méthode de construction de cube à partir d'un flot de données

1 Introduction

Nous l'avons discuté précédemment, permettre une analyse OLAP de l'historique flots de données multidimensionnelles améliorerait la compréhension des données. Remplir cet objectif implique d'être capable de construire efficacement un cube de données alimenté par un flot. Comme cela a été évoqué dans le chapitre 2, une approche, Stream Cube [HCD⁺05], a récemment été proposée pour répondre à cette problématique mais cette approche souffre de quelques limites (*e.g.* un coût de stockage prohibitif, un délai de mise à jour de la structure potentiellement bloquant). Dans ce chapitre, nous étendons cette approche et permettons la construction d'un cube de données compact et dont la mise à jour est extrêmement rapide.

Généralement, l'historique des données de faible granularité n'est consulté que sur un passé récent. Typiquement, il est très rare que des décideurs désirent connaître de manière très précise ce qu'il s'est passé dans le flot il y a dix ans. S'appuyant sur ce constat, nous proposons une technique pour ne matérialiser, sur chaque niveau de précision, que les périodes pendant lesquelles ce niveau est consulté. Cette stratégie permet de réduire significativement la quantité de données stockées pour chaque dimension. L'extension de ce mécanisme au contexte multidimensionnel aboutit à la proposition de *Window Cube*, une structure compacte permettant une analyse multidimensionnelle et multiniveaux efficace.

L'organisation de ce chapitre est la suivante. Au cours de la section 2, nous présentons notre structure, *Window Cube*, pour stocker efficacement un flot de données multidimensionnelles. Dans la section 3, nous proposons une solution pour déterminer automatiquement quelles sont, pour

chaque niveau de granularité de chaque dimension, les portions de l'historique à matérialiser. Ensuite, nous proposons une méthode pour répondre le plus précisément possible aux requêtes utilisateurs dans la section 4. Nous présentons, discutons et comparons les performances obtenues par notre approche sur des jeux de données synthétiques et réelles afin de valider expérimentalement notre proposition. Enfin, nous proposons quelques perspectives associées à cette approche dans la section 6 et développons les plus prometteuses dans le chapitre final de ce manuscrit.

2 Window Cube

Dans cette section, la structure *Window Cube* est définie. Un aperçu général de l'approche est d'abord présenté dans la section 2.1. Nous définissons formellement cette structure au cours de la section 2.2. La section 2.3 décrit le mécanisme de mise à jour de *Window Cube*. Enfin, les algorithmes proposés sont présentés dans la section 2.4.

2.1 Aperçu général de l'approche

Pour satisfaire le critère de compacité, [HCD⁺05] compresse la dimension temporelle grâce aux *tilted time window* présentées dans le chapitre 2 (p. 30). Comme nous l'avons discuté, cette approche nous apparaît insuffisante et nous proposons donc d'étendre cette compression à toutes les dimensions hiérarchisées.

Considérons le tableau 3.1 construit grâce à l'exploitation d'un log de requêtes utilisateurs et qui présente la proportion des fenêtres de la *tilted time window* interrogées pour chaque niveau de granularité des différentes dimensions du cas d'étude. La première ligne de ce tableau s'interprète ainsi : "98% des requêtes utilisateurs impliquant le niveau *Ville* concernent la première fenêtre de la *tilted time window*". L'analyse de ce tableau révèle que le niveau *Vecteur* n'est consulté que sur la première fenêtre de la *tilted time window* alors que, pour le reste de l'historique, les requêtes s'intéressent au niveau *Catégorie vecteur*. Similairement, les requêtes portant sur le niveau *Ville* concernent très majoritairement (*i.e.* 98%) la première fenêtre de la *tilted time window*. Ces requêtes illustrent le fait que *plus l'âge d'une donnée augmente, moins sa précision importe*. L'utilité de conserver durablement l'historique du flot de données à des niveaux de granularité précis est donc discutable.

L'approche s'appuie sur l'idée que les fenêtres de la *tilted time window* devant être matérialisées sur chaque niveau de la hiérarchie d'une dimension devraient tenir compte des habitudes d'interrogation des utilisateurs. Par exemple, un cube stockant les fréquences par *Vecteur* pour le mois courant et généralisant pour le reste de l'historique sur le niveau supérieur (*Catégorie*

Niveau	Intervalle	Fréquence
DIMENSION LIEU		
Ville	W_1	98%
	W_2	2%
Pays	W_1	25%
	W_2	74%
	W_3	1%
Continent	W_4	100%
DIMENSION VECTEUR DE COMMUNICATION		
Vecteur	W_1	100%
Catégorie vecteur	W_2	70%
	W_3	20%
	W_4	10%

TABLE 3.1 – Proportion d’interrogation des niveaux de granularité selon la fenêtre de la tilted time window

vecteur) minimiserait l’espace de stockage tout en garantissant des réponses précises aux requêtes habituelles. Cet exemple illustre à la fois les avantages en terme de compacité d’un tel mécanisme mais aussi et surtout l’importance de définir correctement sur quels niveaux des hiérarchies les différentes parties de l’historique du flot doivent être stockées.

2.2 Formalisation

Dans un premier temps, nous définissons le concept de fonction de précision qui, associée à chaque dimension, permet d’indiquer sur quel niveau de granularité sera stockée chaque fenêtre de la *tilted time window*. Nous combinons ensuite ces fonctions pour déterminer les cuboïdes à matérialiser.

Définition 3.1 (*Fonction de précision*) Soient D_i une dimension, $H_i = (D_i^0, D_i^1, \dots, D_i^{max_i})$ la hiérarchie associée à D_i et $T = \langle W_1, \dots, W_m \rangle$ une tilted time window. La fonction de précision associée à la dimension D_i est définie par $Précision(D_i^j) = \mathcal{T}_j$ telle que :

1. \mathcal{T}_j est une sous-séquence potentiellement vide de fenêtres consécutives de T ;
2. Quel que soit $W_k \in T$, il existe un niveau D_i^l ($0 \leq l \leq max_i$) tel que $W_k \in Précision(D_i^l)$;
3. Si $W_k \in \mathcal{T}_j$, alors il n’existe pas \mathcal{T}_j' tel que $W_k \in \mathcal{T}_j'$;

4. Si $W_k \in \mathcal{T}_j$, alors $W_l \in \mathcal{T}_m$ avec $l > k$ et $m \leq j$.

Exemple 3.1 Les fonctions de précision des dimensions utilisées dans l'exemple sont présentées dans la figure 3.1. Par exemple, $Precision(Pays) = \{W_2, W_3\}$ signifie que (1) si l'on interroge ce niveau sur le dernier mois ou le 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 fréquences en France du dernier mois, il n'est pas possible d'en déduire les ventes sur Montpellier et Paris. Connaissant la distribution des fréquences (e.g. que l'on envoie deux fois plus de SMS à Paris), il serait possible d'approximer les résultats mais sans garantir leur exactitude. (3) A l'inverse, il est tout à fait possible de connaître la quantité exacte de messages envoyés 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.

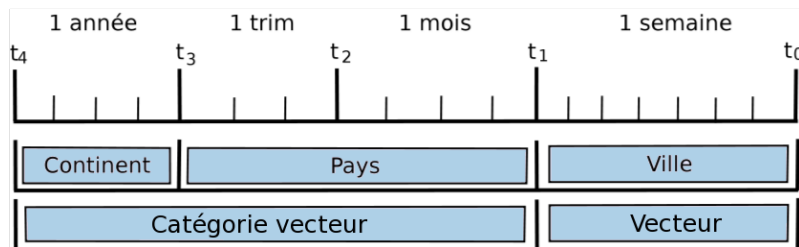


FIGURE 3.1 – Représentation graphique des précisions

Nous introduisons la fonction inverse de *Précision* qui, pour une fenêtre W_j de la *tilted time window* et une dimension D_i , renvoie le niveau associé dans la hiérarchie et la notons :

$$Precision_i^{-1}(W_j) = X$$

avec $W_j \in Precision(X)$ et $k \leq j < l$.

Si l'on reprend les données de l'exemple précédent, nous avons $Precision_{Lieu}^{-1}(W_1) = Ville$.

Ces fonctions de précision définies, nous nous intéressons à leur combinaison via la fonction *Materialize*. Intuitivement, cette fonction prend un cuboïde en entrée et retourne un ensemble de fenêtres de la *tilted time window* tel que chacune de ces fenêtres appartient à la fonction de précision des niveaux de précision des dimensions du cuboïde. Cette fonction est nécessaire pour déterminer quelle partie de l'historique est conservée sur les différents cuboïdes du cube.

Définition 3.2 (*Fonction Materialize*) Soit $C = D_1^{p_1} D_2^{p_2} \dots D_k^{p_k}$ un cuboïde. La fonction *Materialize*(C) est définie ainsi :

$$Materialize(C) = \bigcap_{i=1}^k Precision(D_i^{p_i})$$

Un cuboïde C ne sera matérialisé que si $Materialize(C) \neq \emptyset$. Chaque mesure des cellules de ce cuboïde sera alors une partie d'une *tilted time window* définie par $Materialize(C)$.

Propriété 3.1 *Il n'existe pas de cuboïde $C = D_1^{p_1} D_2^{p_2} \dots D_k^{p_k}$ tel que $Materialize(C) \not\subseteq Precision(D_i^{p_i})$ pour tout $D_i^{p_i} \in D_1^{p_1}, D_2^{p_2}, \dots, D_k^{p_k}$.*

Preuve Cette propriété découle trivialement de la définition de l'intersection. □

Propriété 3.2 *Soient \mathcal{C} un cube de données et T une tilted time window. Nous avons :*

$$\bigcup_{C \in \mathcal{C}} Materialize(C) = T$$

Preuve

1. $T \subseteq \bigcup_{C \in \mathcal{C}} Materialize(C)$

(Par l'absurde) Supposons qu'il existe une fenêtre de T , notée W , qui n'appartienne pas à l'union des $Materialize$ pour tous les cuboïdes de \mathcal{C} . Cela impliquerait que W n'appartienne pas à une fonction de précision d'au moins une dimension du cube. Ceci est impossible par définition d'une fonction de précision (Définition 3.1).

2. $\bigcup_{C \in \mathcal{C}} Materialize(C) \subseteq T$

Selon la définition 3.2, la fonction $Materialize$ est définie comme une intersection ensembliste de fonctions de précision. Par conséquent, l'image de cette fonction est identique à l'image des fonctions de précision, à savoir T . Par définition de l'union ensembliste, l'union de fenêtres de T est un ensemble de fenêtres de T . Par conséquent, l'assertion $\bigcup_{C \in \mathcal{C}} Materialize(C) \subseteq T$ est vraie. □

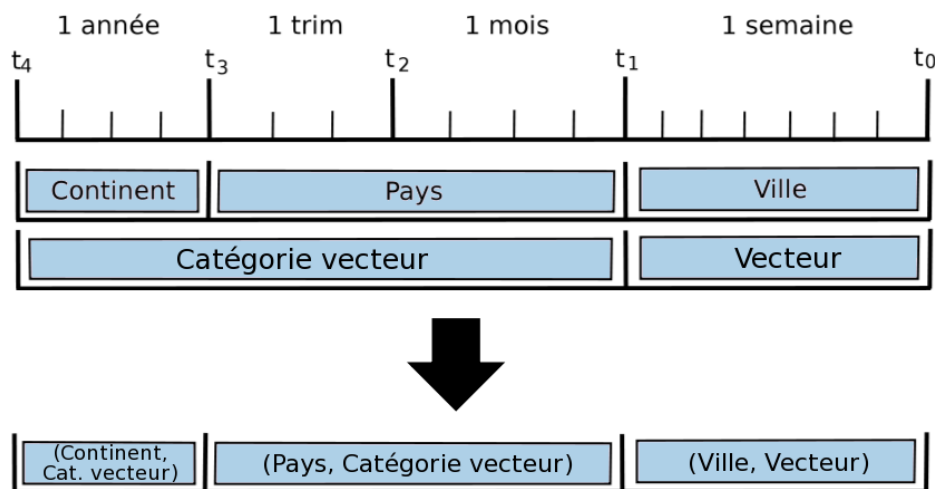


FIGURE 3.2 – Résultat de l'intersection des fonctions de précision

Exemple 3.2 La figure 3.2 illustre le résultat de l'intersection des fonctions de précision des dimensions du cas d'étude. Trois cuboïdes sont matérialisés (Ville, Vecteur) où sera stockée la fenêtre W_1 , (Pays, Catégorie vecteur) contenant les fenêtres W_2 et W_3 et (Continent, Catégorie vecteur) pour W_4 .

Exemple 3.3 La figure 3.3 compare le coût de matérialisation entre Window Cube (figure 3.3(a)) et Stream Cube (figure 3.3(b)). Nous supposons que le popular path utilisé est celui indiqué par les cuboïdes encadrés. Deux remarques peuvent être faites. Premièrement, notre approche engendre la matérialisation de seulement trois cuboïdes (i.e. (Ville, Vecteur), (Pays, Catégorie vecteur) et (Continent, Catégorie vecteur)) alors que, pour Stream Cube, cinq cuboïdes sont matérialisés (i.e. les cuboïdes du popular path). Ensuite, en observant l'intervalle de temps conservé sur chaque cuboïde matérialisé (indiqué sous le nœud du treillis), nous constatons que notre approche conserve bien moins d'éléments que Stream Cube. Par exemple, alors que l'intégralité de la tilted time window est stockée par Stream Cube dans les cellules du cuboïde (Ville, Vecteur), seule la fenêtre W_1 est stockée dans les cellules de ce cuboïde par notre approche.

2.3 Mise à jour

Le modèle proposé étend les *tilted time windows*. Sa mise à jour en reprend le principe de base. La seule différence est que, pour un niveau D_i^p , lorsque l'intervalle de la *tilted time window* défini par $Precision(D_i^p)$ 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

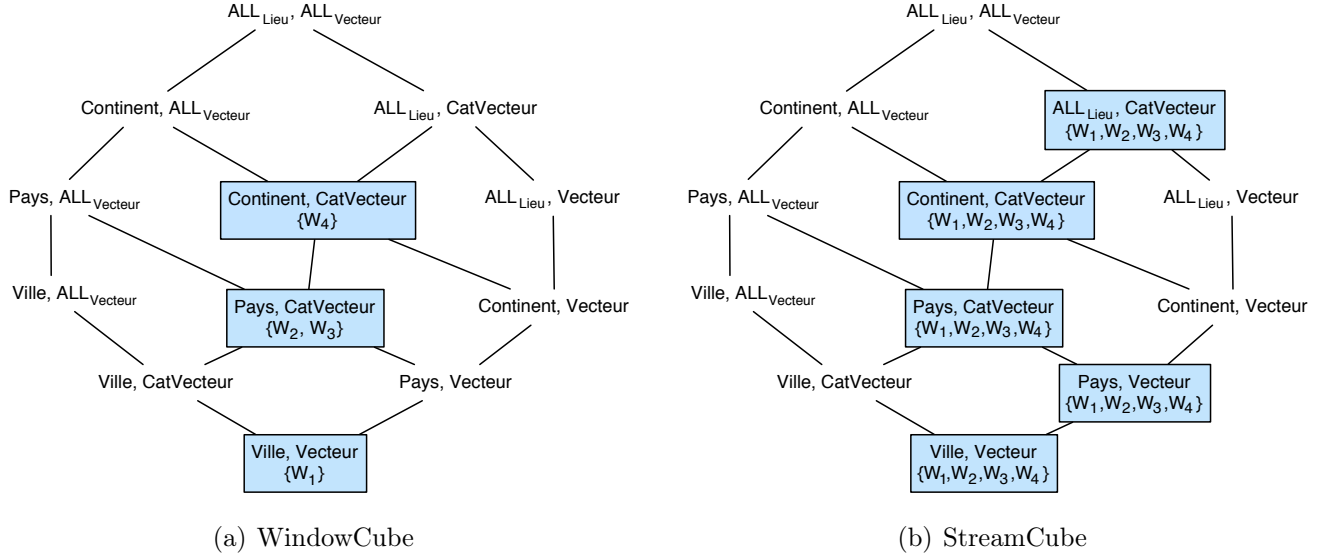


FIGURE 3.3 – Comparaison de la matérialisation entre *Window Cube* et *Stream Cube*

les $(p-1)^\dagger$ 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 $(p-1)^\dagger(c)$.

Exemple 3.4 La séquence à insérer dans la *tilted time window* de la cellule (Montpellier, Tweeter) est $\langle 14, 0, 0, 0, 8, 0, 0, 0, 2, 0 \rangle$. Cette séquence peut se traduire ainsi : il y a eu 14 tweets envoyés dans le premier batch (jour) à Montpellier, aucun le second, ... 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 window* jusqu'à ce qu'elle soit pleine. Ainsi, après le batch 7, nous nous trouvons dans la situation illustrée dans la partie supérieure de la figure 3.4. 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, Tweeter) et 30 pour (Paris, Facebook). La seconde étape consiste à agréger entre elles les valeurs dont la généralisation sur le niveau Pays, Catégorie vecteur conduit au même résultat. Comme $Ville^\sharp(\text{Montpellier}) = Ville^\sharp(\text{Paris}) = \text{France}$ et $CatVecteur^\sharp(\text{Tweeter}) = CatVecteur^\sharp(\text{Facebook}) = \text{Réseaux sociaux}$, 22 est ajouté à 30. Enfin, le résultat est placé dans la première case de la *tilted time window* associée à (France, Réseaux sociaux) et les valeurs du batch 8 (2 et 8) sont insérées dans les *tilted time window* correspondantes. Le résultat produit est illustré dans la partie inférieure de la figure 3.4.

Cet exemple illustre la nécessité de bien déterminer les fonctions de précision pour chaque dimension.

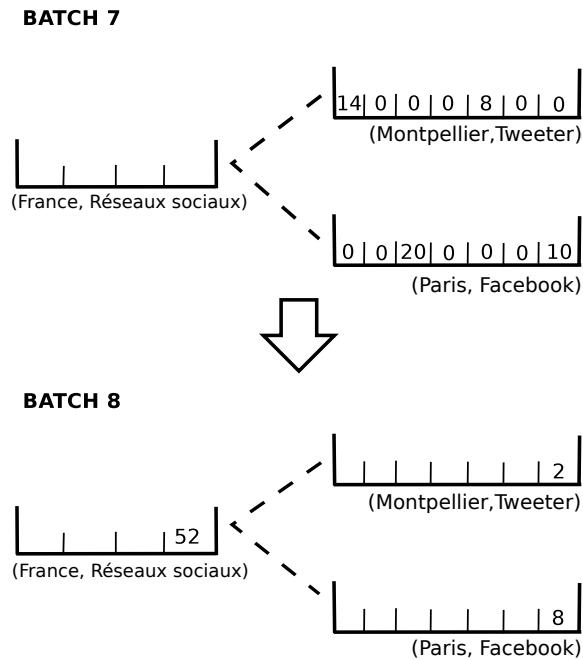


FIGURE 3.4 – Illustration d’un changement de fenêtre

2.4 Algorithmes

Nous présentons ici les algorithmes développés.

L’algorithme général contient deux phases distinctes : l’initialisation et le traitement. La première consiste à (1) récupérer ou calculer les fonctions de précision pour chaque dimension et (2) les combiner. Une fois la structure initialisée, il ne reste plus qu’à traiter les n-uplets arrivant dans le flot sous forme de batch. Il s’agit de l’étape d’insertion et de 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 rapide possible.

L’algorithme 3.1 décrit le fonctionnement général de la méthode et détaille les fonctions appelées pour la phase d’initialisation :

- *precision()* permet de calculer les fonctions de précision présentées dans ce chapitre,
- *intersection()* calcule les cuboïdes à matérialiser et les intervalles de *tilted time window* selon la méthode développée dans la section 2.2

L’algorithme 3.2 décrit les principales étapes de l’insertion d’une mesure *mes* dans une cellule de bas niveau *cell*. Si la cellule n’existe pas, celle-ci est créée (ligne 3) et *mes* est insérée. 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*

Algorithme 3.1: Général

Données : $nbBatches$ le nombre de Batches à générer, $sizeOfBatch$ le nombre de n-uplets dans un batch

```

1 début
  | /* INITIALISATION */
2 | precision() ;
3 | intersection() ;
4 | pour  $i \in 0, 1, \dots, nbBatches - 1$  faire
5 |   | pour chaque  $cell \in BATCH_i$  faire
6 |     | insèreValeur( $cell, mes, 1, i$ );
7 |     | update( $i$ );

```

au début de la fenêtre correspondant (ligne 6). Sinon, une insertion *classique* s'opère. Si la fenêtre n'est pas pleine (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).

Remarque 3.1 *La complexité temporelle de l'algorithme 3.2 est $O(|T|)$ où T est le nombre d'intervalles de la tilted time window. En effet, si l'insertion d'une valeur ne nécessite pas un décalage, cinq opérations élémentaires sont nécessaires. Sinon, le pire des cas serait de propager les valeurs autant de fois qu'il y a de fenêtres dans la tilted time window.*

3 Automatiser la définition des fonctions de précision

Dans la section précédente, nous proposons de résumer un flot de données multidimensionnelles en ne stockant l'historique des données qu'aux niveaux de granularité les plus adaptés. Pour ce faire, nous introduisons les fonctions de précision et les combinons. Une bonne définition de ces fonctions de précision est donc essentielle. Nous consacrons cette section à la description d'une méthode pour automatiser la définition des fonctions de précision en fonction des besoins utilisateurs.

Algorithme 3.2: 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 début
2   si ( $\# cell$ ) alors
3     newCellule(cell,mes,batch) ;
4   sinon
5     si (LastMAJ(cell) = batch) alors
6       cell.window[0] += mes ;
7     sinon
8       si (!full(cell.window)) alors
9         unshift(cell.intervalle,mes);
10        LastMAJ(cell) = numBatch ;
11      sinon
12        si here(cell,window + 1) alors
13          agr = agregation(cell.window);
14          cell.window[0] = mes ;
15          LastMAJ(cell) = batch ;
16          insereValeur(cell,agr,window + 1,batch) ;
17        else
18          agr = agregation(cell.window);
19          cell.window[0] = mes ;
20          cellUp = up(cell,window + 1);
21          LastMAJ(cell) = batch ;
22          insereValeur(cellUp,agr,window + 1,batch) ;

```

3.1 Aperçu général

Le log des requêtes utilisateurs est exploité afin d'évaluer, pour chaque niveau de granularité, quel serait l'impact de matérialiser chaque fenêtre de la *tilted time window*. Nous mesurons cet impact grâce à deux indicateurs :

- *L'imprécision* mesure le nombre de requêtes dont une réponse précise serait impossible si l'on attribue une fenêtre W à un niveau D_i^j (*i.e.* combien de requêtes interrogent W à un niveau de granularité plus fin que D_i^j);

Niveau	Fenêtre	Nombre d'apparitions dans le log
ALL _{Vecteur}	W_1	1
ALL _{Vecteur}	W_2	3
ALL _{Vecteur}	W_3	2
ALL _{Vecteur}	W_4	3
Cat Vecteur	W_1	0
Cat Vecteur	W_2	2
Cat Vecteur	W_3	4
Cat Vecteur	W_4	1
Vecteur	W_1	3
Vecteur	W_2	2
Vecteur	W_3	2
Vecteur	W_4	1

TABLE 3.2 – Log de requêtes

- *L'effort de généralisation* désigne la quantité de requêtes auxquelles une réponse précise et possible grâce à l'opérateur OLAP de généralisation si l'on attribue une fenêtre W à un niveau D_i^j (*i.e.* combien de requêtes interrogent W à un niveau de granularité moins fin que D_i^j).

Bien entendu, attribuer toutes les fenêtres de la *tilted time window* au niveau de granularité le plus fin permettrait de répondre précisément à toutes les requêtes (imprécision nulle) mais contraindrait à un effort de généralisation important. Nous définissons donc une fonction de coût pour quantifier à quel point la matérialisation d'une fenêtre sur un niveau donné respecte ce compromis. Cette fonction de coût permet, pour chaque niveau de trouver la partie de l'historique la plus pertinente à matérialiser sur ce niveau. Puisque l'on désire minimiser ce coût pour tous les niveaux de la dimension considérée, nous transposons ce problème à un problème bien connu et bien traité en théorie des graphes : la recherche du plus court chemin. Nous développons maintenant les étapes de cette méthode.

Exemple 3.5 *Considérons la dimension Vecteur, sa hiérarchie associée (Vecteur, Catégorie vecteur, ALL_{Vecteur}) et la tilted time window utilisée dans ce manuscrit. Le Tableau 3.2 répertorie l'historique des fenêtres interrogées pour chaque niveau de granularité. Une interprétation de la cinquième ligne est "L'historique du flot de données représenté par la fenêtre W_1 n'a jamais été consulté au niveau de granularité CatVecteur".*

3.2 Formalisation

Considérons le tableau 3.3 qui s'appuie sur le log de requêtes présenté dans le tableau 3.2. Pour mieux comprendre sa sémantique, analysons la ligne correspondant au niveau *Vecteur* sur la fenêtre W_2 . La colonne *% satisfiable* indique le pourcentage de requêtes auxquelles il serait possible de répondre si W_2 était matérialisé au niveau *CatVecteur*. Sous cette hypothèse, nous pourrions répondre à $\frac{5}{7}$ des requêtes. Nous obtenons facilement la colonne *% perte* ($1 - \frac{5}{7}$). La dernière colonne, *Effort de généralisation*, comptabilise le pourcentage de requêtes auxquelles il serait possible de répondre en généralisant les données plus précises. Dans l'exemple, si la fenêtre W_2 était matérialisée au niveau *Vecteur*, alors l'effort de généralisation serait $\frac{3}{5}$ puisque 3 requêtes parmi les 5 satisfiables concernent un niveau plus général.

Plus formellement, nous définissons pour une fenêtre W et un niveau n donnés :

- $S_W(n) = \frac{s_W(n)}{Q_W}$ avec $s_W(n)$ le nombre de requêtes auxquelles il est possible de répondre (directement ou avec généralisation) si on matérialisait W au niveau n et Q_W le nombre total de requêtes concernant la fenêtre W (Q est le nombre total de requêtes dans le log),
- $P_W(n) = \frac{p_W(n)}{Q_W}$ avec $p_W(n)$ le nombre de requêtes auxquelles on ne pourrait pas répondre si on matérialisait W au niveau n ,
- $E_W(n) = \frac{g_W(n)}{s_W(n)}$ avec $g_W(n)$ le nombre de requêtes auxquelles on ne pourrait répondre que grâce à une généralisation si on matérialisait W au niveau n .

Intuitivement, pour chaque intervalle t , nous cherchons le niveau qui minimisera à la fois le pourcentage de perte (l'imprécision générée) et l'effort de généralisation. La fonction de coût (dernière colonne du tableau 3.3) est donc définie comme la somme pondérée du taux de perte et de l'effort de généralisation. Formellement, nous avons donc :

$$cout_W(n) = \alpha P_W(n) + (1 - \alpha) S_W(n)$$

Le paramètre α permet de favoriser l'une ou l'autre composante de la fonction de coût. Dans ce chapitre, α est fixé à 0,5 pour que les deux composantes soient d'égale importance. Une étude de la valeur optimale (si elle existe) à donner à ce paramètre peut constituer une perspective de recherche à court terme.

Nous appelons *configuration*, une fonction de précision possible pour une dimension et une *tilted time window* données. Le coût global d'une configuration est la somme des coûts des éléments de cette configuration.

Niveau	% satisfiable ($S_W(n)$)	% perte ($P_W(n)$)	Effort de généralisation ($E_W(n)$)	Coût ($cout_W(N)$)
Fenêtre 1				
3	$\frac{4}{4}$	$\frac{0}{4}$	$\frac{1}{4}$	$\frac{1}{8}$
2	$\frac{1}{4}$	$\frac{3}{4}$	$\frac{1}{1}$	$\frac{7}{8}$
1	$\frac{1}{4}$	$\frac{3}{4}$	$\frac{0}{1}$	$\frac{3}{8}$
Fenêtre 2				
3	$\frac{7}{7}$	$\frac{0}{7}$	$\frac{5}{7}$	$\frac{5}{14}$
2	$\frac{5}{7}$	$\frac{2}{7}$	$\frac{3}{5}$	$\frac{31}{70}$
1	$\frac{3}{7}$	$\frac{4}{7}$	$\frac{0}{3}$	$\frac{4}{14}$
Fenêtre 3				
3	$\frac{8}{8}$	$\frac{0}{8}$	$\frac{6}{8}$	$\frac{3}{8}$
2	$\frac{6}{8}$	$\frac{2}{8}$	$\frac{2}{6}$	$\frac{7}{24}$
1	$\frac{2}{8}$	$\frac{6}{8}$	$\frac{0}{2}$	$\frac{3}{8}$
Fenêtre 4				
3	$\frac{5}{5}$	$\frac{0}{5}$	$\frac{4}{5}$	$\frac{2}{5}$
2	$\frac{4}{5}$	$\frac{1}{5}$	$\frac{3}{4}$	$\frac{19}{40}$
1	$\frac{3}{5}$	$\frac{2}{5}$	$\frac{0}{3}$	$\frac{1}{5}$

TABLE 3.3 – Exploitation du log de requêtes

Définition 3.3 (*Configuration*) Soient T une tilted time window, D_i une dimension et $H_i = \{D_i^0, \dots, D_i^k\}$ la hiérarchie associée à D_i . Une configuration $c = \langle l_1, \dots, l_{|T|} \rangle$ est une séquence de niveaux de granularité de longueur $|T|$ telle que, quel que soit $i \in [1, \dots, |T| - 1]$, l_i désigne un niveau de granularité plus précis ou égal à l_{i+1} . La sémantique associée à chaque membre l_i de cette séquence est : matérialiser la fenêtre W_i au niveau de granularité l_i .

L'imprécision engendrée par une configuration est maîtrisée grâce à l'introduction d'un paramètre utilisateur σ à valeurs entre 0 et 1. Une configuration $c = \langle l_1, \dots, l_{|T|} \rangle$ est alors dite *valide* s'il elle ne contient pas d'élément l_i tel que $P_i(l_i) \geq \sigma$. Sinon, la combinaison est dite *invalid* et n'est plus considérée comme fonction de précision potentielle. Nous prouvons ci-dessous que l'application du paramètre σ sur chaque membre d'une configuration permet de borner l'imprécision totale de la configuration.

Propriété 3.3 Soient $c = \langle l_1, \dots, l_{|T|} \rangle$ et σ un seuil d'imprécision maximale à valeur dans $[0; 1]$. S'il n'existe pas de l_i dans c tel que $P_i(l_i) \geq \sigma$ alors l'imprécision globale de c est elle aussi bornée par σ .

Preuve Soit $\langle l_1, \dots, l_{|T|} \rangle$ une configuration valide. Par définition, nous avons pour chaque l_i

$$P_i(l_i) \leq \sigma,$$

donc,

$$p_i(l_i) \leq \sigma \times Q_i,$$

En sommant les membres de cette inéquation pour toutes les fenêtres, nous obtenons :

$$\sum_{i=1}^{|T|} p_i(l_i) \leq \sum_{i=1}^{|T|} \sigma \times Q_i,$$

Comme $\sum_{i=1}^{|T|} Q_i = Q$, nous obtenons donc :

$$\frac{\sum_{i=1}^{|T|} p_i(l_i)}{Q} \leq \sigma$$

□

Définition 3.4 (*Coût total*) Soient T une tilted time window, D_i une dimension, $H_i = \{D_i^0, \dots, D_i^k\}$ la hiérarchie associée à D_i et $c = \langle l_1, \dots, l_{|T|} \rangle$ une configuration. Le coût total de c est défini comme :

$$\text{coûtTotal}(c) = \sum_{l_i \in c} \text{coût}_{W_i}(l_i)$$

Exemple 3.6 *Par exemple, si l'on considère la configuration $\langle 3, 3, 2, 2 \rangle$, son coût est donc $\frac{1}{8} + \frac{5}{14} + \frac{7}{24} + \frac{19}{40}$ ($\approx 1,248$).*

La recherche de la meilleure fonction de précision équivaut donc à la recherche de la configuration ayant le coût total minimum. Ainsi, le coût total de chaque configuration est calculé et la configuration de coût minimal est élue comme fonction de précision de la dimension concernée. Bien que cette solution oblige à calculer le coût total pour chaque configuration, cette solution est envisageable en pratique en raison du nombre réduit de niveaux de granularité d'une dimension et à la taille restreinte des *tilted time windows* utilisées.

4 Interroger Window Cube

Par définition, les résumés compressent avec perte l'historique d'un flot et ne permettent pas de répondre précisément à toutes les requêtes. Dans notre cas, un niveau de précision p d'une dimension D_i n'est plus consultable après fin_i^p . Nous proposons une méthode d'interrogation qui s'inspire du protocole proposé dans [PJD99].

Dans [PJD99] les données manipulées concernent des diagnostics établis qui peuvent être précis par nature. Par exemple, un diagnostic peut être précis (diabétique dépendant à l'insuline) ou plus vague (diabétique) en fonction des symptômes et examens effectués. Ici, l'imprécision générée est inhérente à l'utilisation des fonctions de précision. Nous sommes néanmoins confrontés aux mêmes problématiques :

1. Comment déterminer efficacement si une réponse précise peut être donnée à une requête ?
2. Sinon (*i.e.* si l'intervalle de temps concerné par la requête n'est plus matérialisé au niveau de précision désiré), quelle stratégie adopter pour fournir malgré tout une réponse pertinente à l'utilisateur ?

Dans un premier temps, nous définissons le type de requêtes considérées ici puis nous décrivons les critères à respecter pour qu'une requête soit satisfiable. Ensuite, nous abordons le cas où une requête n'est pas satisfiable et proposons trois types de réponses alternatives.

4.1 Requêtes considérées et satisfaisabilité

Nous définissons d'abord le type de requêtes que l'on considère dans ce travail.

Définition 3.5 (*Classe de requêtes considérées*) Soient \mathcal{C} un ensemble de critère de sélection de la forme $\langle D_i^j, d_i^j \rangle$ où D_i^j est un niveau de granularité de la dimension D_i et $d_i^j \in \text{Dom}(D_i^j)$, m une mesure du cube considéré (*i.e.* $m \in \mathcal{M}$), f la fonction d'agrégation utilisée pour agréger les

éléments stockés dans les *tilted time windows* et $I = [t_{deb}; t_{fin}]$ un intervalle de temps. La classe de requêtes considérées dans ce travail, \mathcal{Q} est de la forme :

$$\mathcal{Q} = (\mathcal{C}, m, f, I)$$

La sémantique associée à de telles requêtes est : “Quel(le) est le/la f de m tel que \mathcal{C} pendant I ”.

Exemple 3.7 La requête, Q , “Quelles est le nombre total de messages envoyés depuis la France et via des réseaux sociaux cette semaine ?” est transposée ainsi dans le formalisme proposé :

$$Q = (\{\langle \text{CatVecteur}, \text{Réseaux sociaux} \rangle, \langle \text{Pays}, \text{France} \rangle\}, \text{ventes}, \text{somme}, [t_{courant}, t_{courant} - 7])$$

Une requête est satisfiable si le cube contient les informations nécessaires à une réponse précise. Pour déterminer la satisfaisabilité d’une requête, une méthode naïve serait de chercher dans le cube pour déterminer si la réponse précise à la requête est matérialisée. Parcourir la structure n’est pas une solution optimale et il est, de plus, possible de déterminer directement la satisfaisabilité d’une requête en observant la fonction de précision associée à chaque dimension. Nous procédons alors ainsi :

1. Le critère déterminant pour prédire la satisfaisabilité d’une requête est l’intervalle de l’historique interrogé. Nous calculons donc dans un premier temps les fenêtres de la *tilted time window* qui sont concernées par la requête. En supposant que la requête a été formulée à la date t_Q , l’ensemble \mathcal{W}_Q de ces fenêtres est retourné par le résultat de la fonction $\mathcal{W}_Q = \text{Fenêtre}(t_Q, t_{deb}, t_{fin})$.
2. Nous calculons ensuite les fenêtres pour lesquelles une réponse précise à la requête est possible. Pour chaque critère $C_l = \langle D_i^j, d_i^j \rangle$ appartenant à \mathcal{C} , l’ensemble des fenêtres pour lesquelles une réponse précise au niveau D_i^j est possible, noté \mathcal{W}_{C_l} , est donné par :

$$\mathcal{W}_{C_l} = \bigcup_{D_i^k \leq D_i^j} (\text{Précision}^{-1}(D_i^k))$$

\mathcal{W}_{C_l} représente l’union des inverses des fonctions de précision pour les niveaux plus fin ou égaux à D_i^j . L’intersection de ces ensembles indique alors quelles sont les fenêtres de la *tilted time window* pour lesquelles une réponse précise aux critères de sélection contenus dans \mathcal{C} est possible. Cet ensemble, noté $\mathcal{W}_{\mathcal{C}}$ est donc défini par :

$$\mathcal{W}_{\mathcal{C}} = \bigcap_{C_l \in \mathcal{C}} \mathcal{W}_{C_l}$$

3. Ainsi, s’il existe une fenêtre appartenant à \mathcal{W}_Q mais pas à $\mathcal{W}_{\mathcal{C}}$, la requête Q sera considérée *insatisfiable*. Le cas contraire, la requête Q est dite *satisfiable* car il est possible d’y répondre précisément.

Exemple 3.8 *Considérons la requête présentée dans l'exemple 3.7 :*

$$Q = (\{\langle \text{CatProduit}, \text{Consoles} \rangle, \langle \text{Pays}, \text{France} \rangle\}, \text{ventes}, \text{somme}, [t_{\text{courant}}, t_{\text{courant}} - 7])$$

Nous appliquons maintenant les trois étapes décrites ci-dessus pour déterminer si Q est satisfiable.

1. $\mathcal{W}_Q = \text{Fenêtre}(t_{\text{courant}}, t_{\text{courant}} - 7, t_{\text{courant}}) = W_1$

2. *Nous avons :*

$$- \mathcal{W}_{\langle \text{CatProduit}, \text{Consoles} \rangle} = \bigcup_{D_i^k \leq \text{CatProduit}} (\text{Précision}^{-1}(D_i^k)) = \{W_1, W_2, W_3, W_4\}$$

$$- \mathcal{W}_{\langle \text{Pays}, \text{France} \rangle} = \bigcup_{D_i^k \leq \text{France}} (\text{Précision}^{-1}(D_i^k)) = \{W_1, W_2, W_3\}$$

$$- \text{Donc, } \mathcal{W}_C = \mathcal{W}_{\langle \text{CatProduit}, \text{Consoles} \rangle} \cap \mathcal{W}_{\langle \text{Pays}, \text{France} \rangle} = \{W_1, W_2, W_3\}$$

3. *Puisque $\mathcal{W}_Q \subseteq \mathcal{W}_C$, la requête Q est satisfiable.*

Etre capable de déterminer si une requête est satisfiable ou pas n'est pas suffisant. En effet, dans le cas où une requête ne l'est pas, il faut définir une méthode pour "répondre au mieux" à la requête initiale tout en indiquant à l'utilisateur que la réponse sera imprécise. Cette idée de répondre "au mieux" est développée dans la prochaine section.

4.2 Gérer l'imprécision

Nous nous inspirons de [PJD99] et proposons aux utilisateurs des requêtes *alternatives* quand une requête n'est pas satisfiable. Nous présentons maintenant le protocole mis en place.

Si la requête n'est pas satisfiable, une *requête alternative* est proposée à l'utilisateur. Deux approches différentes peuvent être considérées pour proposer une requête alternative :

1. *Privilégier le temps à la granularité* : si une réponse précise à une requête Q n'est pas possible, cela signifie qu'au moins un niveau de précision concerné n'est pas conservé suffisamment longtemps. Ainsi, une requête alternative Q' qui favoriserait le temps au niveau de précision serait une requête telle que (1) l'intervalle de temps de Q et Q' serait identique et (2) les niveaux de précision de Q' serait en adéquation avec l'intervalle de temps concerné par la requête.
2. *Privilégier les niveaux de granularité au temps* : à l'inverse, une solution possible est d'adapter l'intervalle de temps de la requête initiale pour que les niveaux de granularité concernés soient interrogeables avec précision.

La première solution est préférée. En effet, nous supposons que l'intervalle de temps concerné par une requête est un critère fixe pour un utilisateur. Si l'utilisateur accepte la requête alternative, alors celle-ci est exécutée. L'algorithme 3.3 détaille comment est obtenue cette réponse alternative. Dans le cas où l'utilisateur refuse cette requête alternative, il n'est pas non plus souhaitable que

le système ne retourne aucune réponse. Dans une optique coopérative et pour satisfaire au mieux l'utilisateur, nous proposons de retourner l'ensemble des réponses *proches* de sa requête initiale. Nous appelons cette réponse *verbose answer*. L'algorithme 3.4 présente la méthode pour parvenir à un tel résultat. Son principe est le suivant : pour chaque intervalle concerné par la requête initiale, il faut d'abord trouver le cuboïde stockant cet intervalle puis exécuter une requête intermédiaire concernant cet intervalle et les niveaux de granularité du cuboïde.

Algorithme 3.3: RequêteAlternative

Données : Une requête non satisfiable $Q = (\mathcal{C}, m, f, I)$, les fonctions de précision associées aux dimensions dans Q , t la date où à laquelle a été formulée Q

Résultat : $Q_{alt} = (\mathcal{C}', m, f, I)$

```

1 début
2    $\mathcal{C}' \leftarrow \emptyset$  ;
   /* Récupération de la fenêtre qui représente la plus vieille partie de
      l'historique dans  $\mathcal{W}_Q$  */
3    $\mathcal{W}_Q \leftarrow Fen\hat{e}tre(t, t_{deb}, t_{fin})$  ;
4    $W_{old} \leftarrow max(\mathcal{W}_Q)$  ;
   /* Pour chaque critère de sélection dans  $\mathcal{C}$  */
5   pour chaque  $\langle D_i^j, d_i^j \rangle \in \mathcal{C}$  faire
       /* Si la précision demandée est trop fine */
6       si  $D_i^j > Precision_{D_i}^{-1}(W_{old})$  alors
7            $D_i^{j'} \leftarrow Precision_{D_i}^{-1}(W_{old})$  ;
8            $d_i^{j'} \leftarrow (D_i^{j'})^\#(d_i^j)$  ;
9            $\mathcal{C}' \leftarrow \mathcal{C}' \cup \langle D_i^{j'}, d_i^{j'} \rangle$  ;
10          sinon
11               $D_i^{j'} \leftarrow D_i^j$  ;
12               $d_i^{j'} \leftarrow d_i^j$  ;
13               $\mathcal{C}' \leftarrow \mathcal{C}' \cup \langle D_i^{j'}, d_i^{j'} \rangle$  ;
       /* Composition de la requête alternative */
14    $Q_{alt} \leftarrow (\mathcal{C}', m, f, I)$  ;
15   retourner  $Q_{alt}$  ;

```

Exemple 3.9 Considérons $Q_2 =$ “Quel est le total de messages envoyés via Tweeter depuis Montpellier pendant la période totale de l'historique du flot?”. La requête alternative serait $Q'_2 =$ “Quel

Algorithme 3.4: ReponseImprécise

Data : Une requête non satisfiable $Q = (C', m, f, I)$, les fonctions de précision associées, t
la date de formulation de Q

Result : \mathcal{Q} un ensemble de requêtes

```

1 begin
2    $\mathcal{W}_Q \leftarrow Fen\hat{e}tre(t, t_{deb}, t_{fin})$ ;
   /* Pour chaque  $W_k$  dans  $\mathcal{W}_Q$  */
3   foreach  $W_k \in \mathcal{W}_Q$  do
4      $\mathcal{C}_j = \bigcup_{\langle D_i^j, d_i^j \rangle} \langle Precision_{D_i}^{-1}(W_k), (Precision_{D_i}^{-1}(W_k))_{D_i}^\#(d_i^j) \rangle$ 
5
6    $Ens_Q \leftarrow Ens_Q \cup (\mathcal{C}_j, m, f, I)$ 
7 return  $Ens_Q$ ;

```

est le total de messages envoyés via les réseaux sociaux en Europe pendant la période totale de l'historique du flot?”. En cas de refus, le système retournerait la réponse verbale suivante :

- Le total de messages Tweeter envoyés depuis Montpellier sur W_1
- Le total de messages envoyés depuis la France via un réseau social sur W_2 et W_3
- Le total de messages envoyés depuis l'Europe via un réseau social sur W_4

5 Expérimentations

Nous évaluons maintenant l'approche proposée dans ce chapitre. Pour cela, nous adaptons une démarche en deux temps. Dans un premier temps, nous nous comparons à Stream Cube [HCD⁺05]. Nous réalisons ensuite une étude de la consommation mémoire et du temps de mise à jour de la structure sur des jeux de données synthétiques et réels. Pour ce faire, l'influence de plusieurs paramètres (nombre de dimensions, degré et profondeur des hiérarchies et fonctions de précision) est étudiée.

Deux conditions sont indispensables pour valider notre approche :

1. Une consommation mémoire bornée,
2. Un délai de mise à jour de la structure borné.

Les jeux de test synthétiques proviennent d'un générateur de batches de données multidimensionnelles aléatoires satisfaisant certaines contraintes (nombre de dimensions, degré des hiérarchies,

etc.). Nous pouvons ainsi évaluer précisément 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 2GHz avec 2Go de mémoire vive sous Linux 2.6.20. Les méthodes (Window Cube et Stream Cube) ont été implémentées en Perl version 5 et utilisent une base de données MySQL. Nous utilisons une fenêtre temporelle logarithmique de taille 8 ($\log_2(250) = 7,96\dots$).

Parmi les critiques formulées à l'encontre de Stream Cube, une d'elles concerne sa matérialisation excessive. Nous examinons alors le nombre de fenêtres matérialisées par les deux approches sur des jeux de données synthétiques. Ensuite, la sous-section 5.2 présente et discute les résultats obtenus sur différents jeux de données synthétiques. La mémoire vive occupée et le temps d'insertion d'un batch dans Window Cube démontrent que notre structure est bien adaptée au contexte des flots. Ces bonnes propriétés établies, nous appliquons notre méthode sur des données issues de sondes réseaux. Les résultats obtenus montrent qu'une telle structure serait avantageuse pour un administrateur réseau.

5.1 Comparaison avec Stream Cube

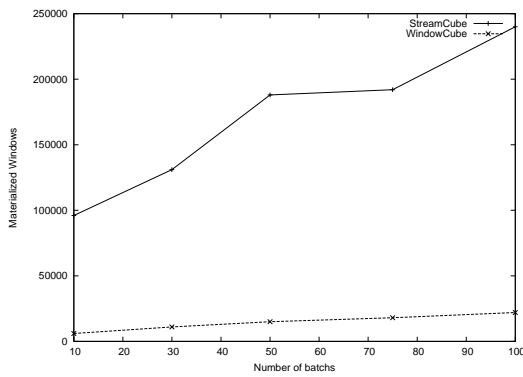
Au cours de cette série d'expérimentations, nous comparons le nombre de fenêtres matérialisées sur des jeux de données synthétiques entre *Stream Cube* et *Window Cube*.

La figure 3.5(a) présente le nombre de fenêtres matérialisées en fonction du nombre de batchs. Dans cette série de tests, la longueur du *popular path* et le nombre de cuboïdes matérialisés par WindowCube sont égaux à 6. Le jeu de test utilisé est D5C5L3T1. Nous remarquons alors que le treillis est de hauteur 16. Les résultats obtenus sont intéressants puisqu'en moyenne, notre approche matérialise 10 fois moins de fenêtres que Stream Cube. En outre, l'impact du nombre de batchs sur les fenêtres matérialisées est faible. A l'inverse, le nombre de batchs impacte grandement sur la matérialisation de Stream Cube. Ainsi, puisqu'un flot est une séquence potentiellement infinie de batchs, notre approche semble particulièrement adaptée par rapport à Stream Cube.

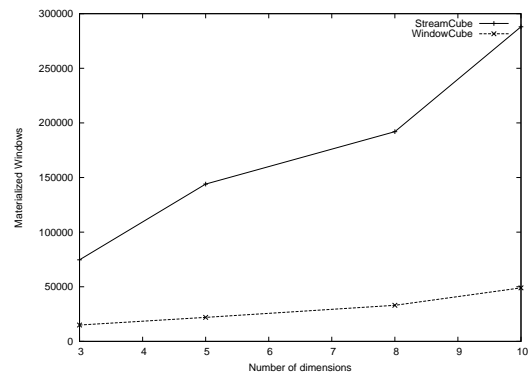
La figure 3.5(b) présente le nombre de fenêtres matérialisées en fonction du nombre de dimensions d'un n-uplet. De même que dans la précédente série de tests, la longueur du *popular path* et le nombre de cuboïdes matérialisés sont équivalents. Ainsi, pour 10 dimensions, notre approche matérialise 11 cuboïdes. Nous notons que le treillis est de hauteur 31. Le jeu de test utilisé est C5L3B50T1. Les résultats obtenus sont très satisfaisants. En effet, notre approche matérialise entre 4 et 6 fois moins de fenêtres que Stream Cube. Là aussi, les courbes présentées suivent un

comportement similaire au précédent test. Ainsi, le nombre de dimensions du jeu de données a peu d'influence sur les performances de notre approche. A l'inverse, ce paramètre dégrade considérablement les performances de Stream Cube en terme de matérialisation.

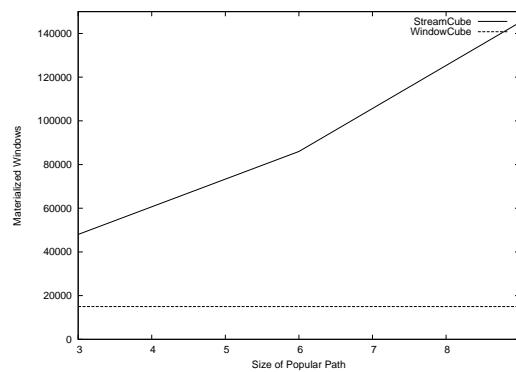
Dans les précédents tests, nous fixons la longueur du *popular path* pour qu'elle soit équivalente au nombre de batchs matérialisés par notre approche. Cette considération avait pour objectif de ne pas biaiser les résultats. Cependant, dans le cas où le treillis est de taille 30, un *popular path* de longueur 10 est peut être insuffisant. En effet, il est très possible que celui-ci soit plus long. Ainsi, nous testons le nombre de fenêtres matérialisées en fonction de la longueur du *popular path*. Puisque, pour notre approche, le nombre de cuboïdes matérialisés est dépendant des fonctions de précision et qu'elles restent identiques, le nombre de fenêtres stockées par notre approche reste inchangé. Le jeu de test utilisé est D5C5L3B50T1. Nous notons que notre approche matérialise 6 cuboïdes. La figure 3.5(c) présente les résultats obtenus. Même quand la longueur du *popular path* est de longueur 3, Stream Cube matérialise deux fois plus de fenêtres que notre proposition. Ce rapport passe à 7 quand la longueur du *popular path* atteint 9.



(a) Nb de batchs (D5C5L3T1)



(b) Nb de dimensions (C5L3B50T1)



(c) Longueur du popular path (C5D5B50T1)

FIGURE 3.5 – Comparaison du nombre de fenêtres matérialisées par rapport à Stream Cube

Les chapitres précédents supposaient que notre approche était moins coûteuse que Stream Cube. Dans cette section, nous le démontrons empiriquement. En effet, les résultats obtenus attestent du réel bénéfice de notre proposition. Dans la prochaine section, nous évaluons notre structure sur des jeux de données synthétiques difficiles.

5.2 Résultats sur des jeux de données synthétiques

Dans cette section, nous évaluons notre approche sur des jeux de données synthétiques en faisant varier de nombreuses caractéristiques du flot. Ainsi, nous déterminons quels sont les paramètres impactant le plus notre approche et analysons le comportement de notre structure face à des conditions extrêmes. Dans la section précédente, nous avons vu que le nombre de fenêtres stockées est faible comparé à Stream Cube. Nous nous intéressons maintenant à deux autres paramètres critiques dans les flots de données : la mémoire vive occupée et le temps d'insertion d'un batch dans WindowCube. Les conditions d'évaluation sont identiques à la section précédente.

Intervalles	<i>Expensive</i>	<i>Slim</i>	<i>Balanced</i>
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			$A_3B_2C_2D_3E_3$
t_2, t_3		$A_2B_2C_2D_3E_2$	
t_3, t_4		$A_1B_2C_2D_3E_1$	
t_4, t_5		$A_1B_1C_1D_2E_1$	
t_5, t_6		$A_1B_1C_1D_1E_1$	
t_6, t_7		$A_1B_1C_1D_1E_1$	$A_1B_1C_1D_1E_1$
t_7, t_8			

TABLE 3.4 – Cuboïdes matérialisés

Dans cette première série d'expérimentations nous étudions leur impact sur les performances de notre proposition. Les configurations testées sont présentées dans le tableau 3.4. La première (*expensive*) conserve l'essentiel de l'historique sur le cuboïde le plus précis. À 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. Le jeu de test utilisé est D5L3C5B250T20.

Les figures 3.6 et 3.7 présentent les résultats obtenus. Les temps de traitement sont exposés dans les figures 3.6(b), 3.6(a) et 3.7(a). Les courbes obtenues suivent un comportement similaire. Premièrement, le temps de traitement augmente légèrement pendant les premiers batchs à cause

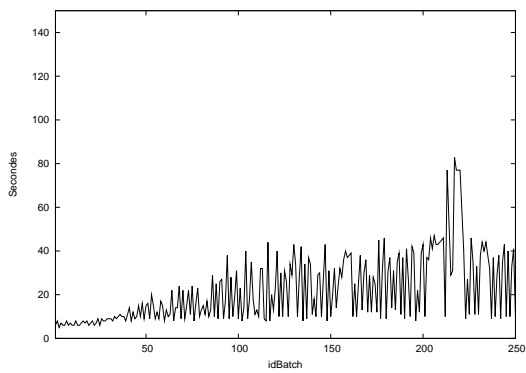
du faible nombre de cellules présentes dans la structure. Nous observons ensuite une alternance entre des temps très rapides et d'autres plus longs. Les pics correspondent aux batchs où un changement de fenêtre s'effectue. Enfin, le temps maximal est borné à environ 60 secondes. Nous notons également 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 3.7(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 utilisé est L3C5B250T20. Les fonctions de précision utilisées sont équilibrées (proches de la configuration *Balanced*). Les figures 3.8 et 3.9 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.

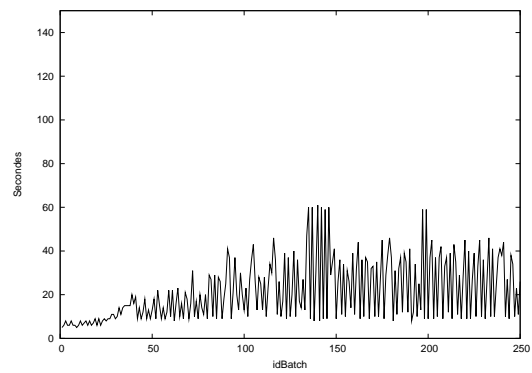
L'influence du nombre de dimensions est plus important que celle des fonctions de précision mais reste malgré tout limitée. De plus, même avec 15 dimensions, le temps de traitement ne dépasse pas une minute.

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 3.10 et 3.11 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 résultats sont présentés sur les figures 3.12 et 3.13. Le temps de traitement suit le même comportement que pour les autres expérimentations. La consommation de mémoire est encore stable à 10Mo.

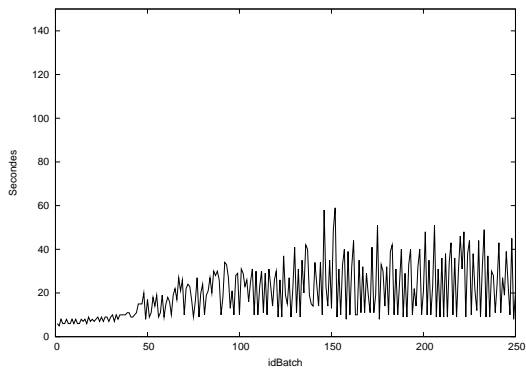


(a) Temps vs *Expensive* (D5L3C5B250T20)

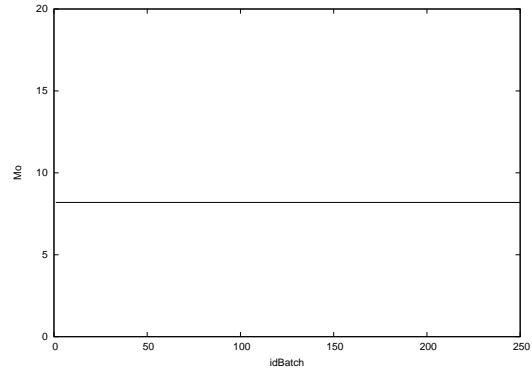


(b) Temps vs *Slim* (D5L3C5B250T20)

FIGURE 3.6 – Influence des fonctions de précision (1/2)

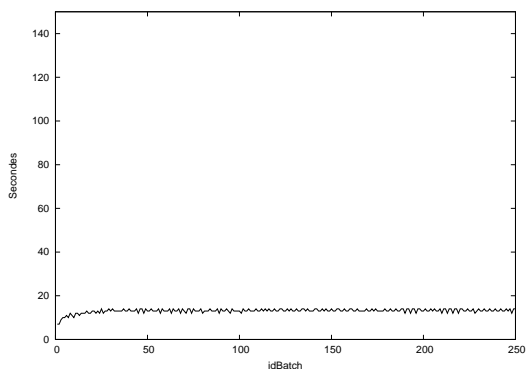


(a) Temps vs *Balanced* (D5L3C5B250T20)

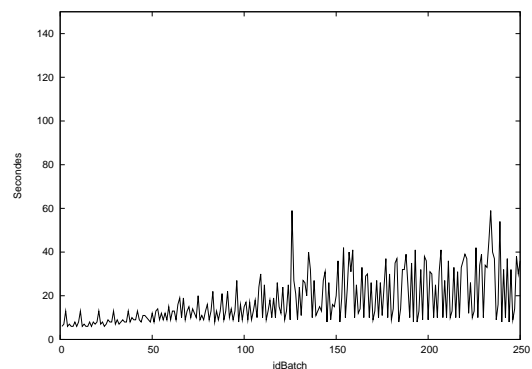


(b) Mémoire vive consommée (D5L3C5B250T20)

FIGURE 3.7 – Influence des fonctions de précision (2/2)

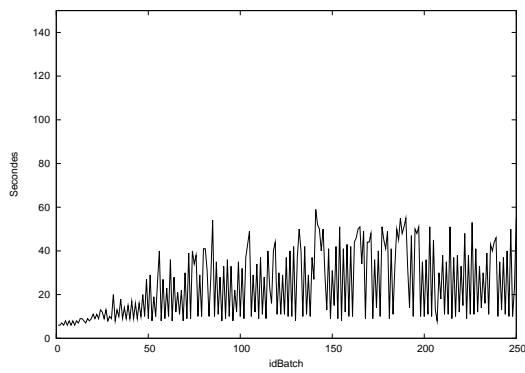


(a) Temps vs 2 dimensions (L3C5B250T20)

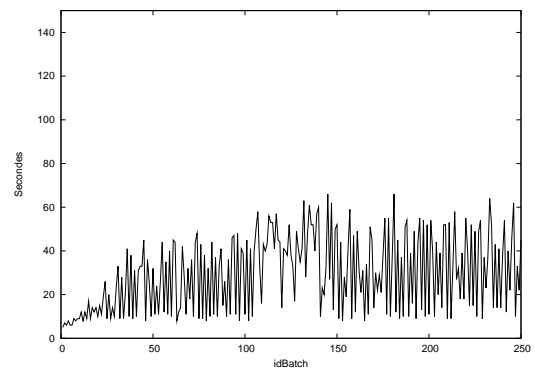


(b) Temps vs 5 dimensions (L3C5B250T20)

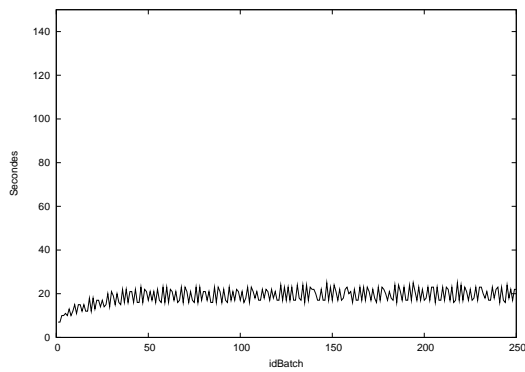
FIGURE 3.8 – Influence du nombre de dimensions (1/2)



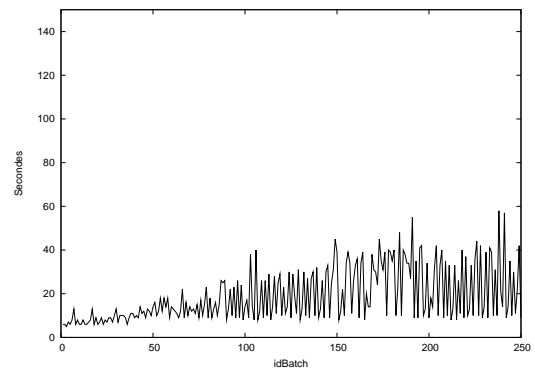
(a) Temps vs 10 dimensions (L3C5B250T20)



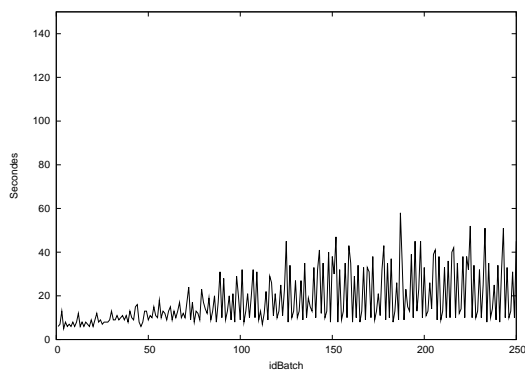
(b) Temps vs 15 dimensions (L3C5B250T20)

FIGURE 3.9 – Influence du nombre de dimensions (2/2)

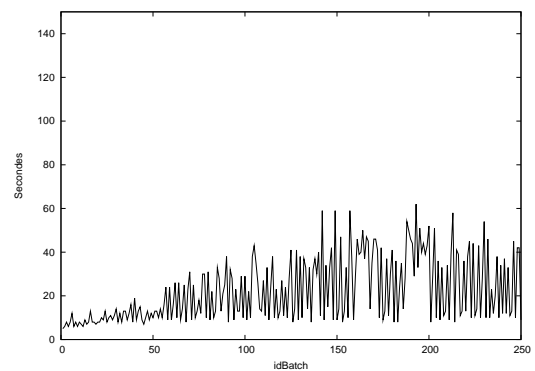
(a) Temps vs degré 2 (D5L3B250T20)



(b) Temps vs degré 5 (D5L3B250T20)

FIGURE 3.10 – Influence du degré des hiérarchies (1/2)

(a) Temps vs degré 10 (D5L3B250T20)



(b) Temps vs degré 20 (D5L3B250T20)

FIGURE 3.11 – Influence du degré des hiérarchies (2/2)

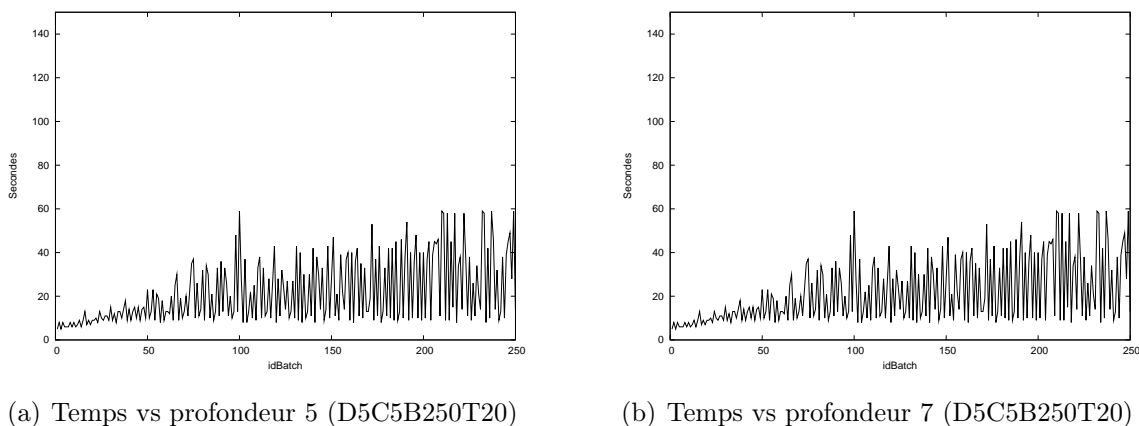


FIGURE 3.13 – Influence de la profondeur des hiérarchies (2/2)

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. Cette série d'expérimentations permet de conclure quant au respect de notre approche aux critères de mise à jour rapide et de compacité.

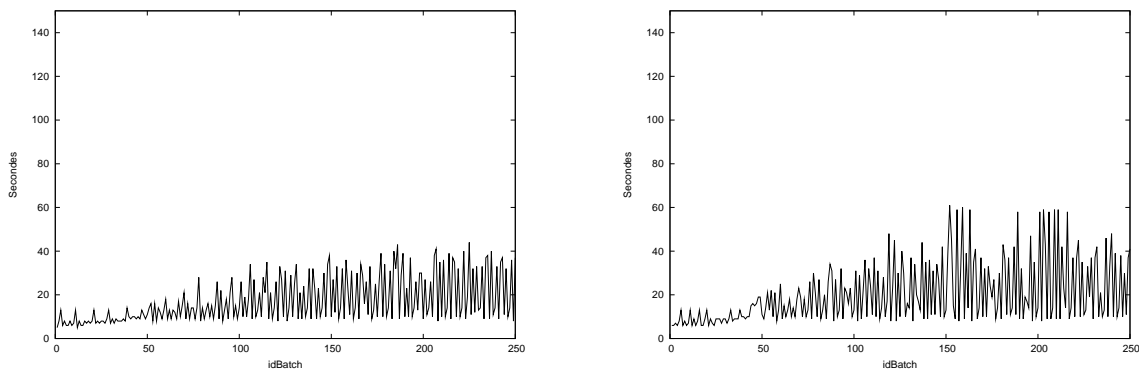
5.3 Application sur des données réelles

Les deux sections précédentes permettent de conclure que (1) notre approche est moins coûteuse que StreamCube et que (2), même dans des conditions extrêmes notre approche obtient de bons résultats. Nous testons maintenant notre structure sur des données réelles issues de sondes réseaux. Ce jeu de données a un intérêt certain puisque l'analyse de réseau est un domaine sensible. En effet, que cela soit pour optimiser le trafic, déterminer les habitudes des utilisateurs ou détecter des attaques, il est nécessaire de disposer d'outils d'analyse offrant une bonne réactivité à l'administrateur. Le jeu de données contient 204 batchs de 20 K n-uplets. Un n-uplet est décrit sur 13 dimensions. Ces dimensions sont extraites des entêtes des trames TCP et des datagrammes IP. Elles ont été choisies par un expert pour leur pertinence (port source, port destinataire, TTL, etc.). La figure 3.14 (resp. figure 3.15) illustre la structure d'une trame TCP (resp. d'un datagramme IP). La moitié des dimensions sont hiérarchisées (profondeur 2 ou 3). Par exemple, les ports de destination supérieurs à 32771 sont généralisés en *multimédia* car ces ports sont utilisés par des applications de streaming vidéo et audio.

La figure 3.16 illustre les résultats obtenus sur ce jeu de données. Nous constatons que (1) le temps d’insertion d’un batch dans la structure est borné à 15 secondes et (2) que la mémoire vive consommée reste constante à 8 Mo. Ces bonnes performances permettent d’envisager la mise en place d’un tel système pour la surveillance d’un réseau. Ainsi, un administrateur pourra, par exemple, consulter les plages d’adresses IP atteintes la semaine précédente. Dans le cas d’une attaque par déni de service sur un serveur, l’administrateur doit réagir rapidement. Il n’est donc pas utile de stocker l’historique des adresses destinataires sur six mois. Ainsi, en ne conservant l’historique des adresses IP atteintes que sur l’heure en cours, le gain de matérialisation sera conséquent et il sera malgré tout possible d’exhiber des accès anormalement élevés à un serveur (*i.e.* détecter une éventuelle attaque par déni de service). Notre structure permettrait de minimiser l’espace occupé par l’historique tout en garantissant la réactivité de l’administrateur réseau.

6 Discussion

Dans ce chapitre, nous proposons Window Cube, une structure de données compacte permettant une interrogation multi-niveaux de l’historique du flot. Les fonctions de précision permettent (1) de ne pas conserver les fenêtres de la *tilted time window* qui sont rarement consultés et (2) de minimiser la redondance en ne matérialisant pas ce qui peut être obtenu par généralisation. La combinaison de ces fonctions forme une structure de données compacte aux propriétés intéressantes. Cependant, une bonne définition de ces fonctions de précision est cruciale pour ne pas générer une trop grande imprécision du résumé. Pour cela, nous proposons une méthode de définition automatique de ces fonctions de précision. Nous exploitons ainsi les informations stockées dans le log de requêtes et définissons une fonction de coût pour quantifier le compromis entre l’imprécision et l’effort de généralisation. Nous prouvons que les fonctions de précision automa-



(a) Temps vs profondeur 3 (D5C5B250T20)

(b) Temps vs profondeur 4 (D5C5B250T20)

FIGURE 3.12 – Influence de la profondeur des hiérarchies (1/2)

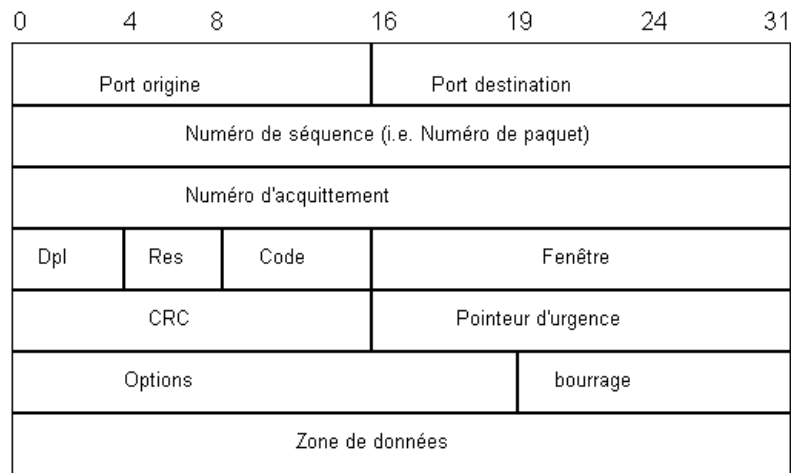


FIGURE 3.14 – Structure d'une trame TCP

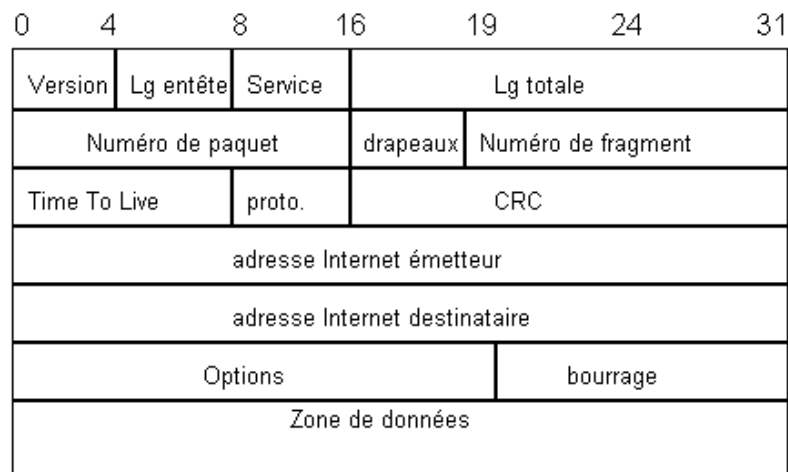


FIGURE 3.15 – Structure d'un datagramme IP

tiquement définies ne dépassent pas un seuil d'imprécision spécifié par l'utilisateur. Enfin, afin pour pallier l'imprécision inhérente à une telle structure, nous nous appuyons sur les travaux de [PJD99] pour fournir une réponse *acceptable* à une requête dont une réponse précise est impossible.

De plus, l'approche proposée dans ce chapitre peut être appliquée à de nombreux autres domaines d'application. A titre d'exemple, nous avons capturé le flot des messages Twitter envoyés depuis les principaux pays anglophones au cours de ces 6 derniers mois et avons défini des fonctions de précision pour quelques dimensions de ce flot (*e.g.* lieu d'émission). Nous présentons quelques diagrammes réalisés pour visualiser l'historique du nombre de messages Tweeter. Par souci de visibilité, ces diagrammes ne représentent l'historique de ce flot que sur la dimension *Lieu d'émission*. La figure 3.17 représente le nombre de tweets envoyés par ville pendant le mois en cours. L'historique de ce nombre de messages envoyés lors des deux précédents mois (figure 3.18) est conservé

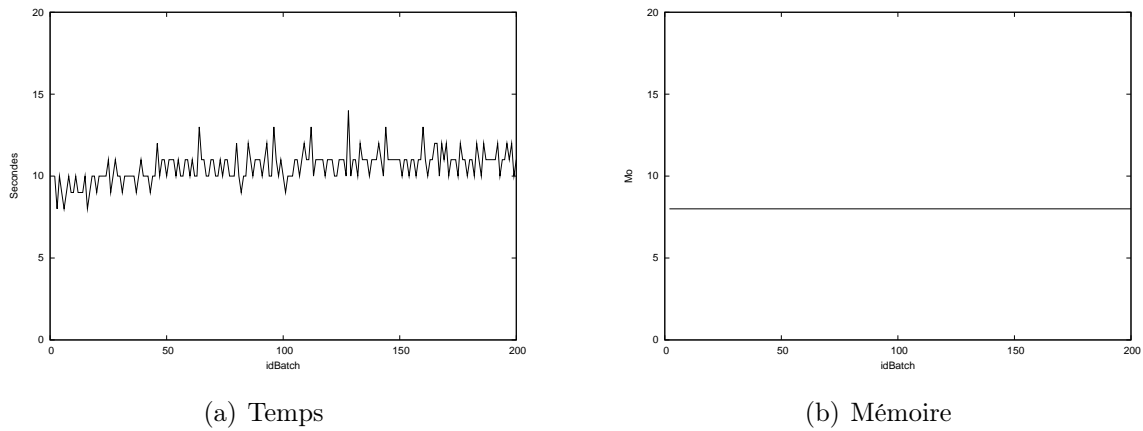


FIGURE 3.16 – Résultats obtenus sur des données réelles

au niveau supérieur (*i.e.* *état* si la ville se situe aux Etats-Unis ou *pays* sinon). Enfin, l'historique plus ancien n'est conservé qu'au niveau *Pays* (figure 3.19).

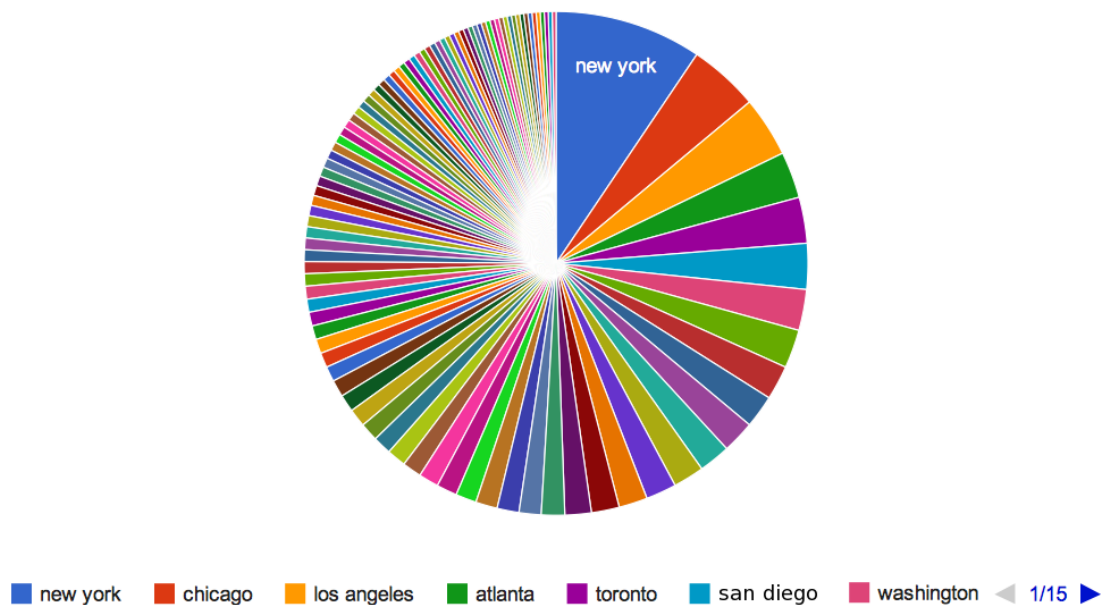


FIGURE 3.17 – Nombre de tweets du mois courant conservé par ville

Les perspectives associées à ce travail sont nombreuses. Nous en détaillons ici quelques unes. Tout d'abord, nous avons supposé qu'un log de requêtes utilisateur existe et nous nous appuyons dessus pour définir automatiquement les fonctions de précision. Si une telle connaissance n'existe pas *a priori*, ces fonctions devront être définies manuellement par le(s) utilisateur(s). En outre, si les besoins utilisateurs évoluent au cours du temps (*i.e.* si l'imprécision générée par un ensemble

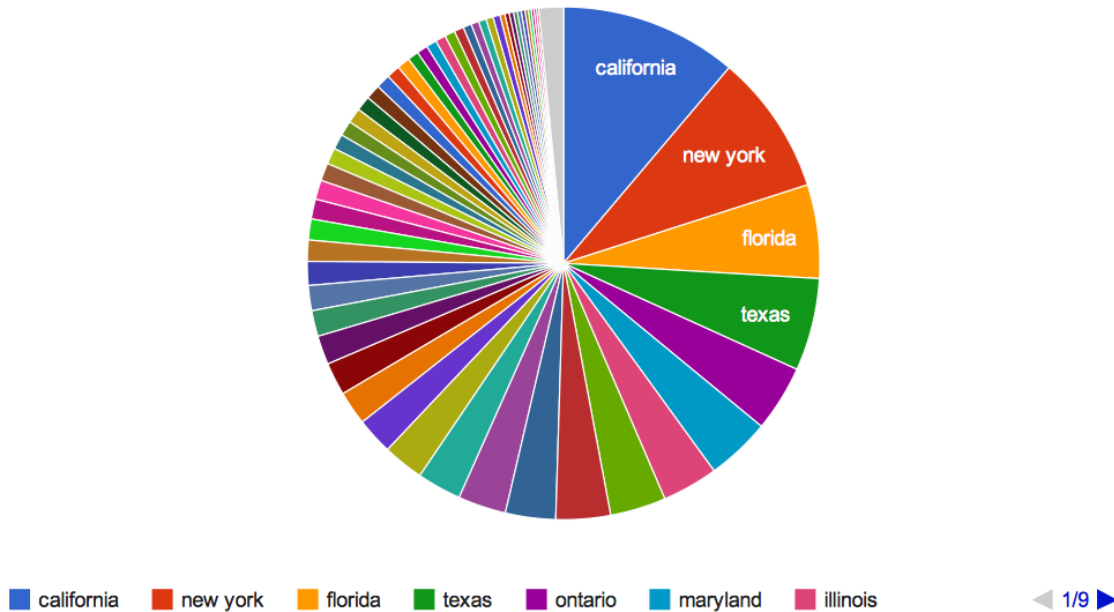
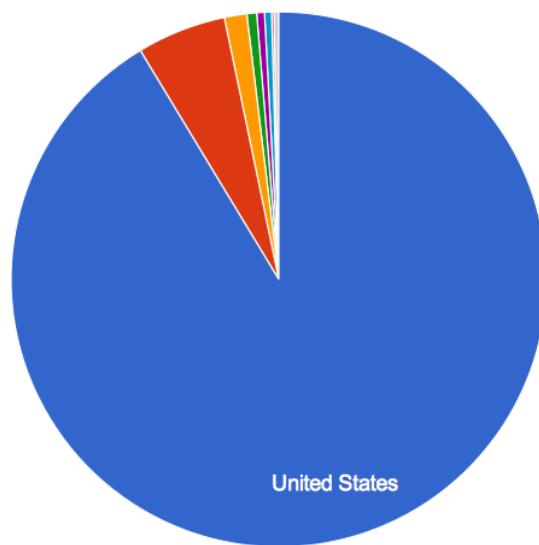


FIGURE 3.18 – Nombre de tweets des deux derniers mois conservé par état ou pays

de nouvelles requêtes utilisateur dépasse le seuil σ), il sera nécessaire de recalculer à la volée de nouvelles fonctions de précision. Une solution pour éviter ce calcul systématique est d'utiliser un mécanisme similaire à [GG07] lorsque la distribution des données évolue et qu'un nouveau modèle doit être appris. Il s'agirait alors dans notre cas de conserver l'historique des ensembles de fonctions de précision. Ainsi, l'imprécision générée par les fonctions de précision courantes en fonction des nouvelles requêtes utilisateur serait périodiquement calculée. Si cette imprécision dépasse le seuil utilisateur σ , une recherche serait lancée sur les ensembles de fonctions de précision précédents. Si un ensemble satisfait la contrainte de l'imprécision, ces nouvelles fonctions sont appliquées. Si aucun ensemble ne satisfait le seuil d'imprécision un nouvel ensemble de fonctions sera alors calculé.

Ensuite, il serait pertinent de considérer une gamme plus large de requêtes (*range query* [HAMS97], requêtes *top k* [AW00], ...) que celles considérées dans ce chapitre afin de permettre une plus grande flexibilité d'analyse.

Enfin, nous apportons dans ce chapitre une solution pour pallier les deux premières limites de Stream Cube (*i.e.* matérialisation redondante et propagation coûteuse en temps). Malgré tout, l'approche proposée dans ce chapitre ne permet pas de pallier la limite associée à l'utilisation des *tilted time windows* quant à l'agrégation systématique et aveugle des données. Tel est l'objectif du prochain chapitre où nous proposons une structure de liste dynamique remplaçant certaines *tilted time windows* dans la conservation de l'historique d'une cellule du cube. Le résumé ainsi produit satisfera mieux la contrainte de représentativité.



■ United States ■ Canada ■ Greenland ■ Cuba ■ Honduras ■ Mexico ■ Jamaica ■ Other

FIGURE 3.19 – Nombre de tweets du semestre dernier conservé par pays

Chapitre 4

Prise en compte de la distribution variable des données dans un flot

1 Introduction

Dans le chapitre précédent, nous proposons une méthode pour construire un cube de données alimenté par un flot de données multidimensionnelles. En optimisant à la fois la taille de la structure aux besoins utilisateurs et le temps de sa mise à jour, nous pallions deux limites de [HCD⁺05] (*i.e.* occupation mémoire potentiellement importante et mise à jour pénalisante). Dans ce chapitre, nous souhaitons remédier à la limite liée à l'utilisation des *tilted time windows* (décrite dans la section 5.2 du chapitre 2) pour compresser la dimension temporelle au sein d'une cellule d'un cube de données. En effet, le mécanisme d'agrégation étant automatique, une telle structure n'est pas appropriée pour capturer fidèlement l'historique d'un item apparaissant occasionnellement dans le flot. Pour cela, il est nécessaire de proposer à la fois une structure de résumé et un mécanisme d'agrégation associé qui considère le contenu de cette structure pour la compresser plus justement. Dans ce chapitre, nous proposons une telle structure et répondons à la question “*comment résumer finement l'historique d'un item apparaissant occasionnellement dans un flot tout en respectant les conditions inhérentes à son traitement (occupation mémoire bornée et mise à jour rapide) ?*”.

En outre, grâce à un travail mené avec des psychologues cognitivistes, nous avons été amenés à nous intéresser à une thématique de recherche présentant de nombreuses similarités avec le résumé de flot de données multidimensionnelles : le fonctionnement de la mémoire humaine. Via ses cinq sens, l'Homme doit quotidiennement faire face à une multitude d'informations précises, imprévisibles, arrivant rapidement et auxquelles on peut généralement associer une hiérarchie de concepts. En ce sens, cette masse d'informations possède de nombreuses caractéristiques communes avec un flot de données multidimensionnelles. De plus, un individu est incapable de retenir l'intégralité des

informations qu’il reçoit continuellement et doit donc sélectionner l’information pertinente, l’agréger et mettre en place des mécanismes d’oubli. La mémoire humaine peut donc être considérée comme un système très efficace de résumé de flots de données. Dès lors, il est légitime de se poser les questions suivantes : “*Comment les processus de mémorisation et d’oubli fonctionnent-ils ? Certaines théories issues de la psychologie cognitive peuvent-elles être adaptées dans le cadre de ce travail ?*”. Il est apparu qu’une théorie actuelle sur le fonctionnement de la mémoire humaine pouvait être exploitable dans notre contexte. En effet, il semblerait que l’humain mémorise les informations par association de concepts. Cette théorie, connue sous le nom de *mémoire constructive* implique que les souvenirs ne seraient pas mémorisés indépendamment mais seraient plutôt associés les uns aux autres formant ainsi un graphe de concepts. Par exemple, le souvenir d’un voyage à l’étranger renvoie aux souvenirs du trajet, des paysages vus, des personnes rencontrées, de l’hôtel, . . . D’une certaine manière, la sémantique des itemsets fréquents présente des similarités avec la théorie de la *mémoire constructive*. Effectivement, un itemset fréquent représente justement une association fréquente entre items (concepts). Permettre de synthétiser l’historique de ces itemsets fréquents offre des capacités d’analyse intéressantes pour un décideur qui disposerait d’un outil pour mesurer l’évolution des relations entre concepts.

Dans ce chapitre, nous proposons une structure de résumé de flots d’éléments plus ou moins complexes et qui pallie les limites des *tilted time windows*. Pour cela, nous utilisons une structure de graphe possédant différentes catégories de nœuds et proposons de conserver l’historique de certains éléments dans des listes dynamiques. Le mécanisme d’agrégation associé à une liste prend en compte la proximité temporelle des éléments stockés et permet ainsi la conservation durable et précise des apparitions occasionnelles de l’élément représenté.

L’organisation de ce chapitre est la suivante. Au cours de la section 2, nous présentons un aperçu général de la structure proposée. Dans la section 3, nous décrivons en détail le mécanisme de liste proposée ainsi que les opérations associées. Ensuite, nous montrons dans la section 4, comment cette structure permet le résumé de données brutes issues d’un flot de données multidimensionnelles. La section 5 montre comment cette structure peut être utilisée pour synthétiser les itemsets fréquents du flot de données. Nous présentons et discutons ensuite les performances obtenues par notre approche sur des jeux de données synthétiques et réelles pour valider expérimentalement la méthode proposée dans ce chapitre. Enfin, nous discutons des perspectives associées à cette approche dans la section 6 et en développerons certaines dans la conclusion générale de ce manuscrit .

2 Aperçu général de l'approche

Parmi les objectifs à atteindre dans ce chapitre, permettre de résumer pareillement différents types d'informations sur le flot (*e.g.* les données brutes ou les itemsets fréquents de ce flot) rend impossible l'utilisation d'une structure de cube de données traditionnelle telle qu'utilisée dans le chapitre précédent. Dès lors, pour gérer un contenu hétérogène tel que les hiérarchies associées aux dimensions, les items multidimensionnels ou encore les itemsets, nous adoptons une structure de graphe dont les nœuds stockeront diverses informations en fonction de leur rôle.

Initialement, ce graphe n'est composé que des hiérarchies des différentes dimensions. Ces nœuds structurels et figés ne stockent aucune information mais sont essentiels dans le processus d'interrogation. Au fur et à mesure que les *objets* arrivent, des nœuds stockant le résumé à différents niveaux de granularité sont créés, mis à jour ou supprimés. Pour pallier le manque de représentativité des *tilted time windows*, l'historique des éléments bruts du flot est conservé dans une liste dynamique. A chaque apparition d'un *objet* X dans un batch, l'estampille temporelle du batch est ajoutée dans la liste représentant X . Une fusion d'éléments dans la liste n'est réalisée que si X apparaît dans des batchs proches. Les nœuds représentant des *objets* plus généraux conservent l'historique des apparitions de ces objets dans des *tilted time windows*. La figure 4.1 présente un aperçu de la structure proposée appliquée aux données décrites dans le cas d'étude. Nous considérons que les objets à résumer sont les données brutes du flot. Les membres des différentes hiérarchies (*e.g.* *Paris*, *Montpellier*) sont modélisés par des nœuds structurels (*NS* sur la figure) qui servent de points d'ancrage à deux catégories de nœuds : les *nœuds de bas niveau* (*NBN* sur la figure) qui conservent l'historique des items les plus précis et les *nœuds généralisés* (*NG* sur la figure) qui conservent l'historique à un niveau de précision plus général. Nous présentons maintenant la structure de liste proposée et les mécanismes associés pour contrôler sa taille.

3 Listes dynamiques : définitions et fonctionnement

Au cours de cette section, nous présentons le fonctionnement des listes indépendamment de l'objet dont elle représente l'historique des apparitions. Pour ce faire, nous considérons que tout élément d'un batch B_i est de la forme (Id, val) où Id est l'identifiant d'un objet (*e.g.* un item multidimensionnel ou un itemset fréquent du batch) et val représente une mesure associée à cet objet (*e.g.* le nombre d'apparitions de Id dans le batch B_i).

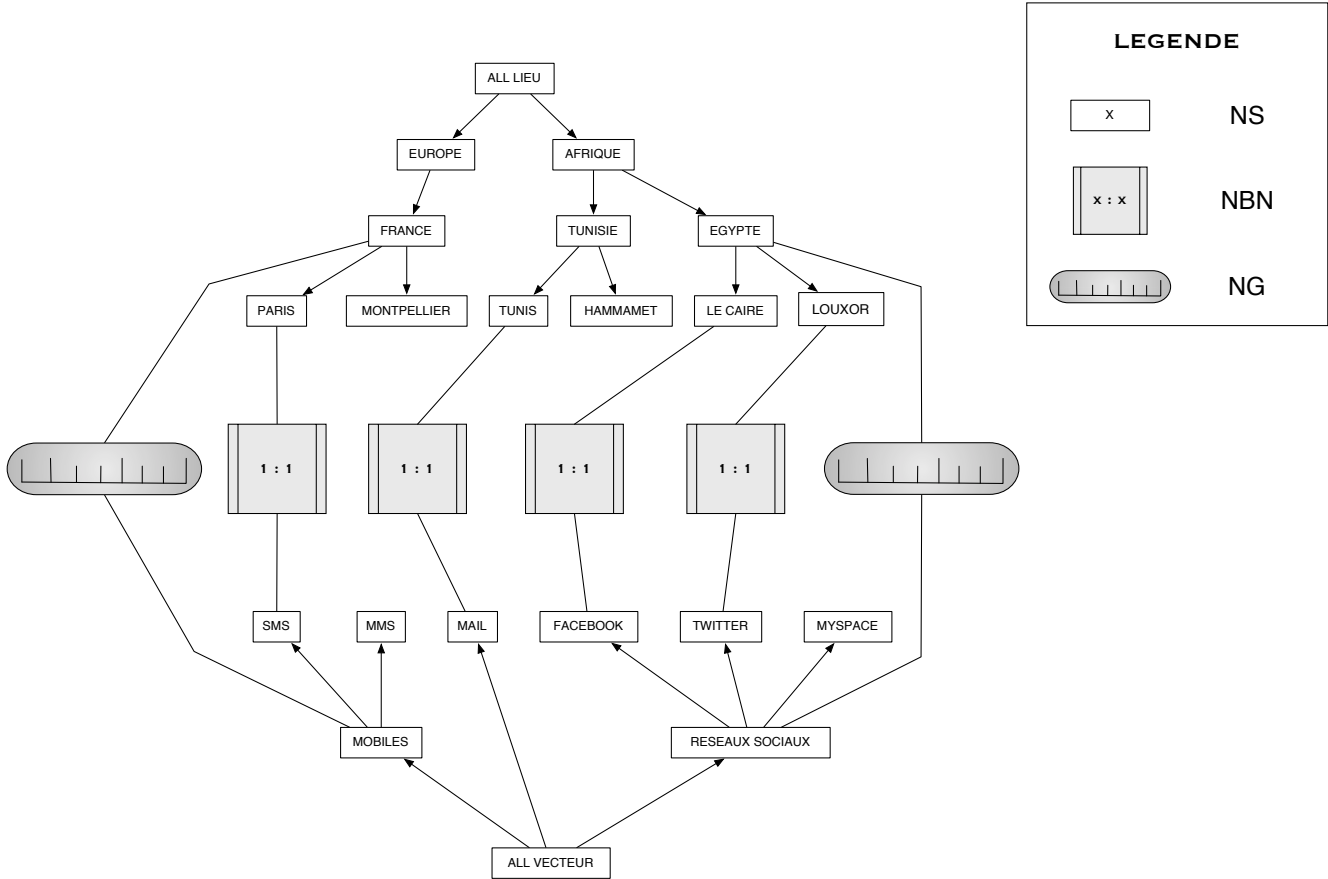


FIGURE 4.1 – Illustration de la structure proposée.

3.1 Définitions

Définition 4.1 (Élément d'une liste) Soit $F = B_0, B_1, \dots$ une séquence potentiellement infinie de batchs. Une liste associée à un objet Id est une suite d'éléments E_1, \dots, E_k de la forme $E_i = (deb_i, fin_i, val_i)$ tel que deb_i et fin_i identifient deux batchs, $deb_i \leq fin_i$ et $val_i \in \mathbb{R}$.

Définition 4.2 (Fonctions de manipulation des éléments) Soit $E = (deb, fin, val)$ un élément. Nous introduisons les fonctions et notations suivantes pour manipuler cet élément :

- $Deb(E)$ désigne le début de l'intervalle de temps concerné par E (i.e. $Deb(E) = deb$) ;
- $Fin(E)$ désigne la fin de l'intervalle de temps concerné par E (i.e. $Fin(E) = fin$) ;
- $Val(E)$ désigne la valeur stockée dans E (i.e. $Val(E) = val$) ;
- $isIn(t, E)$ indique si l'identifiant du batch t fait parti de l'intervalle de temps concerné par E . Plus formellement :

$$isIn(t, E) = \begin{cases} 1 & \text{si } deb \leq t \leq fin \\ 0 & \text{sinon} \end{cases}$$

Définition 4.3 (Liste dynamique) Soient $F = B_0, B_1, \dots$ une séquence potentiellement infinies de batchs, Id un identifiant et $\lambda = (E_1, \dots, E_k)$ une liste d'éléments telle qu'il n'existe aucun iden-

tifiant de batch, t , concerné par deux éléments, E et E' de λ (i.e. il n'existe pas $\{E, E'\} \subseteq \lambda$ tel que $isIn(t, E) = 1$ et $isIn(t, E') = 1$). Une liste dynamique $L = (Id, \lambda)$ représente ainsi l'historique des apparitions distinctes de Id dans F .

Définition 4.4 (Fonctions de manipulation des listes) Soit $L = (Id, \lambda)$ une liste. Nous introduisons les fonctions et notations suivantes pour manipuler cette liste :

- $Id(L)$ désigne l'identifiant de la liste L (i.e. $Id(L) = Id$),
- $Liste(L)$ désigne la liste associée à L (i.e. $Liste(L) = \lambda$),
- $Elmt(L, t)$ retourne, s'il existe, l'élément E de la liste λ tel que $isIn(t, E) = 1$ et l'ensemble vide sinon. Plus formellement :

$$Elmt(L, t) = \begin{cases} E & \text{si } \exists E = (deb, fin, val) \in L \text{ tel que } isIn(t, E) = 1 \\ \emptyset & \text{sinon} \end{cases}$$

- $Elmts(t_1, t_2, L)$ désigne l'ensemble des éléments de λ concernés par l'intervalle de temps $[t_1, \dots, t_2]$. Plus formellement :

$$Elmts(t_1, t_2, L) = \bigcup_{t \in [t_1, \dots, t_2]} Elmt(t, L)$$

- $Add(E, L)$ insère l'élément E dans L (i.e. $\lambda = \lambda \cup \{E\}$)
- $Suppr(E, L)$ supprime l'élément E (i.e. $\lambda = \lambda - \{E\}$)

3.2 Fonctionnement des listes dynamiques

Remplissage

Le principe de mise à jour des listes est le suivant. Soit $F = (B_1, B_2, \dots)$ un flot de données découpé en batchs. Pour chaque couple (O, val) d'un batch B_i , l'élément $E = (i, i, val)$ est inséré dans la liste associée à l'objet O (i.e. $Add(E, L)$ avec $L = (O, \lambda)$). Malgré tout, cette solution n'est pas satisfaisante. En effet, la présence d'un objet O dans tous les batchs du flot entraînerait une liste de taille non bornée pour représenter les apparitions de O dans le flot de données. La nécessité de satisfaire le critère de compacité impose la mise en place de mécanismes de contrôle et de gestion de la taille d'une liste que nous décrivons ci-dessous.

Borner la taille des listes

Le mécanisme de fusion d'éléments d'une liste s'appuie sur le principe suivant : des éléments d'une liste seront fusionnés entre eux si l'objet identifiant cette liste apparaît dans des batchs

proches les uns des autres. Cette notion subjective de proximité sera définie, justifiée et discutée dans la suite de chapitre. Pour l’instant, nous considérons que cette proximité est simplement représentée par un paramètre δ à valeur dans \mathbb{N} . Plus formellement, considérons une liste $L = (Id, \lambda)$ telle que $\lambda = (E_1, \dots, E_n)$. S’il existe une sous-liste $\lambda' = (E_l, \dots, E_m)$ (avec $1 \leq l < m \leq n$) telle que $\lambda' \subseteq \lambda$ et $|Deb(E_l) - Fin(E_m)| \leq \delta$ alors les éléments de λ' seront fusionnés entre eux. Le nouvel élément E_{fus} résultant de cette fusion est défini ainsi : $E_{fus} = (Deb(E_l), Fin(E_m), agr(Val(E_l), \dots, Val(E_m)))$ où *agr* est une fonction d’agrégation définie par l’utilisateur¹. Dans le cas où un élément peut participer à deux fusions, la fusion faisant intervenir les éléments les plus vieux est privilégiée. Dans la suite de ce chapitre, nous utiliserons la moyenne arithmétique comme fonction d’agrégation. Intuitivement, un tel mécanisme pallie le manque de représentativité des *titled time windows* en regroupant les éléments d’une liste représentant des batchs temporellement proches tout en garantissant que les éléments éloignés ne seront pas fusionnés. Ce mécanisme est illustré ci-dessous.

Exemple 4.1 *Considérons la liste présentée dans la figure 4.2(a) et supposons $\delta = 3$. Dans cette liste, trois fusions sont possibles :*

- Fusionner E_1, E_2 et E_3 ,
- Fusionner E_4 et E_5 ,
- Fusionner E_5 et E_6 .

Parmi ces trois fusions possibles, seulement deux peuvent être réalisées. En effet, les sous-listes $\lambda_1 = (E_4, E_5)$ et $\lambda_2 = (E_5, E_6)$ respectent la condition de proximité mais λ_1 contient des éléments plus anciens que λ_2 . Le résultat du mécanisme de fusion sur cette liste est présenté dans la figure 4.2(b).

Malgré tout, ce mécanisme de fusion ne permet pas de borner la taille d’une liste. Cela s’explique par l’impossibilité de prédire les données circulant dans un flot et donc, le nombre de fusions qui seront réalisées. L’introduction d’un paramètre utilisateur, T_MAX , à valeur dans \mathbb{N} permet de maîtriser la taille d’une liste. Il est important de préciser que ce paramètre n’indique pas exactement la taille maximale d’une liste. En effet, dans la mesure où le $T_MAX^{ème}$ élément de la liste pourra éventuellement participer à une fusion, il est nécessaire d’attendre δ batchs pour vérifier si cette fusion se produira. Ainsi, en combinant le mécanisme de fusion et ce paramètre utilisateur, le nombre maximum d’éléments stockés dans une liste est $T_MAX + \delta$. Lorsque la liste a atteint sa taille maximale, l’élément le plus ancien de la liste est supprimé. Notons ici, qu’il n’y a aucune contrainte sur la durée de conservation d’un élément dans une liste. Ainsi, si un objet O apparaît

1. Aucune propriété d’additivité ou de semi-additivité n’est requise pour cette fonction.

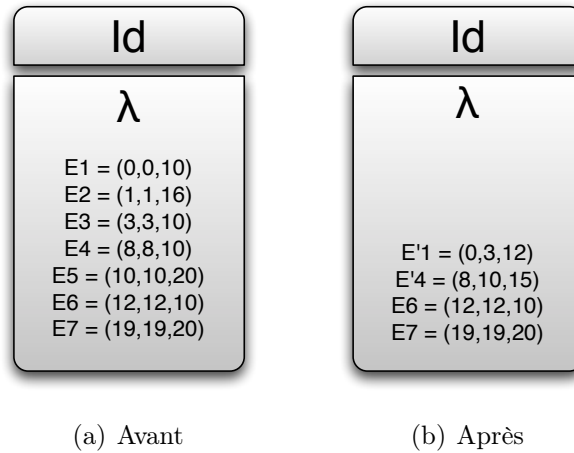


FIGURE 4.2 – Illustration du mécanisme de fusion au sein d'une liste dynamique

très rarement dans le flot de données, l'historique de ses apparitions sera conservée durablement.

La structure de liste et les mécanismes associés étant présentés, nous nous intéressons maintenant à l'utilisation de ces listes dans un contexte de résumé de données brutes.

4 Résumer un flot de données multidimensionnelles brutes

Une donnée brute du flot de données est un $(n+1)$ -uplet de la forme (d_1, \dots, d_n, m) où les d_i appartiennent au domaine de définition de $D_i^{max_i}$ (*i.e.* la plus faible granularité de la dimension D_i) et m est une mesure associée à $d_1 \times \dots \times d_n$. Un élément dont les apparitions seront résumées est de la forme $E = (Id, val)$ telle que $Id = (d_1, \dots, d_n)$ et $val = m$. Nous décrivons maintenant les différents types de nœuds composant le graphe.

Les nœuds structurels (NS)

Un nœud structurel est un nœud dans lequel ne sera stocké aucun historique. Ici, ces nœuds sont exclusivement composés des hiérarchies associées aux différentes dimensions D_1, \dots, D_n . Leur fonction est de servir d'ancrage à la structure pour faciliter l'interrogation du résumé à différents niveaux de granularité.

Les nœuds de bas niveau (*NBN*)

Un nœud de bas niveau stocke l'historique des apparitions d'une donnée brute du flot de données grâce à une liste dynamique. Lorsqu'une fusion est réalisée dans cette liste, le résultat de l'agrégation est propagé le long des nœuds représentant les généralisations de cet élément.

Les nœuds généralisés (*NG*)

Un nœud généralisé représente l'historique d'une généralisation d'une donnée brute dans une *tilted time window*. L'utilisation de ce mécanisme ne contredit le critère de représentativité grâce à l'utilisation de listes dynamiques pour résumer le flot au plus fin niveau de granularité. Pour déterminer quelles généralisations sont matérialisées, nous reprenons l'idée du *popular path* proposé dans [HCD⁺05] et le notons $\mathbf{P} = C_1, \dots, C_k$ (avec C_1 le cuboïde de base).

Les interactions entre ces nœuds de nature différente sont développées ci-dessous.

4.1 Mise à jour de la structure

Limitations concernant la fusion d'éléments dans une liste

Dans la section précédente, nous avons proposé un mécanisme pour fusionner les éléments d'une liste. Ce mécanisme est basé sur la proximité temporelle des éléments d'une liste telle que k éléments d'une liste $L = (Id, \lambda)$ seront fusionnés si l'intervalle de temps représenté par ces éléments ne dépasse pas δ , un seuil fixé par l'utilisateur. Cependant, puisque le résultat de cette fusion est propagé dans des nœuds généralisés représentant l'historique des généralisations de Id , certaines restrictions sur la valeur prise par le paramètre δ sont nécessaires.

Considérons $T = \langle W_1, \dots, W_k \rangle$ le modèle de *tilted time window* utilisée dans les nœuds généralisés. Une définition arbitraire du paramètre δ conduirait à des erreurs lors de la propagation de la valeur agrégée dans les nœuds généralisés correspondants. En effet, l'intervalle de temps représenté par la fusion de ces éléments pourrait correspondre à différentes granularités de T rendant impossible l'insertion de cet élément fusionné dans les nœuds généralisés. Cette remarque justifie de restreindre le paramètre δ à correspondre à une granularité temporelle de T (*i.e.* $\delta = Repr(W_2), \dots$ ou $\delta = Repr(W_k)$).

En outre, pour une fenêtre *éloignée*², W_e , $Repr(W_e)$ est très grand. Il n'est alors pas réaliste de conserver $T_MAX + Repr(W_e)$ éléments pour attendre une éventuelle fusion. Il est alors

2. Nous entendons par fenêtre *éloignée*, une fenêtre conservant l'historique très ancien du flot.

nécessaire de fixer une fenêtre maximale $W_{max} \in \{W_2, \dots, W_k\}$ pour limiter la taille d'une liste et ainsi prévenir toute explosion mémoire. W_{max} est un paramètre utilisateur défini en fonction de l'espace mémoire disponible et du souhait de l'utilisateur de conserver durablement ou pas l'historique précis des apparitions des éléments bruts du flot. Ainsi, le nombre maximal d'éléments pouvant être stocké dans une liste est $T_MAX + Repr(W_{max})$.

Propagation d'un élément fusionné

Supposons que l'élément $E = (deb, fin, val)$ soit le résultat d'une fusion d'éléments au sein de la liste $L = (Id, \lambda)$. Afin de propager correctement cet élément dans la *tilted time window* correspondant à la généralisation de Id sur C_2 , il est nécessaire de rechercher les nœuds de bas niveaux dont l'identifiant possède la même généralisation que Id sur C_2 . Les listes associées à ces nœuds sont parcourues pour rechercher, \mathcal{M} , l'ensemble des éléments n'ayant jamais participé à une propagation dont l'intervalle de temps correspondant est compris entre deb et fin . Il est en effet essentiel de ne pas utiliser un même élément dans plusieurs propagation sous peine de fausser le résumé produit. Pour éviter cela, chaque élément de \mathcal{M} est marqué et ne pourra plus participer à aucune propagation. Les valeurs de ces éléments sont ensuite agrégées entre elles au moyen de la fonction d'agrégation utilisée dans le mécanisme de fusion. Le résultat de cette agrégation, val_{agr} , est alors inséré dans la *tilted time window* du nœud correspondant à la généralisation de Id sur C_2 . La propagation aux *titled time windows* plus générales est réalisée grâce au mécanisme classique des *titled time windows*.

Mise à jour d'un nœud de bas niveau

Soit $L = (Id, \lambda)$ une liste stockée dans un nœud de bas niveau N . L'algorithme général d'insertion d'un élément $E = (deb, fin, val)$ peut être décrit de la manière suivante. Dans un premier temps, la taille de λ est évaluée et comparée à T_MAX . Si la taille de λ est plus petite, l'élément est inséré et une recherche de fusion est lancée. Si une fusion est possible, celle-ci est déclenchée. Si par contre, la liste excède T_{max} éléments, nous notons $E_{max} = (deb_{max}, fin_{max}, val_{max})$ le $T_{max}^{ième}$ élément de λ et évaluons si $fin - deb_{max} > Repr(W_{max})$. Si tel est le cas, cela signifie qu'aucune fusion ne sera possible entre E_{last} et E . Par conséquent, l'élément le plus ancien de la liste est supprimé. Dans le cas contraire, E est ajouté car il pourra éventuellement participer à une fusion dans le futur. L'algorithme 4.1 récapitule ce processus.

Algorithme 4.1: Mise à jour

Données : $c = (deb, fin, val)$ l'élément à insérer, $L = (id, \lambda)$ une liste, W_{max}

```

1 si  $taille(\lambda) < T\_MAX$  alors
2   si  $(\exists l : (deb_i, fin_i, val_i) \in liste\ tq\ (fin - deb_i) \in \{Repr(W_1), \dots, Repr(W_{max})\})$  alors
3     Fusion des éléments de  $I$  à  $c$  dans  $\lambda$ ;
4   sinon
5     Ajout de  $c$  en queue de  $\lambda$ ;
6 sinon
7    $E_{max} = (deb_{max}, fin_{max}, val_{max})$  le  $T_{max}^{ième}$  élément de  $\lambda$ ;
8   si  $(fin - deb_{max}) \in \{Repr(W_1), \dots, Repr(W_{max})\}$  alors
9     Fusion des éléments de  $E_{max}$  à  $c$  dans  $\lambda$ ;
10    ;
11   sinon
12     si  $(fin - deb_{max}) < Repr(W_{max})$  alors
13       Ajout de  $c$  en queue de  $\lambda$ ;
14     sinon
15       Suppression du plus ancien élément de  $\lambda$ ;

```

5 Résumer des itemsets fréquents

Cette section décrit les modifications à apporter à la structure proposée pour permettre le résumé d'itemsets extraits sur les batchs du flot. Dans un premier temps, nous justifions la pertinence de pouvoir construire un tel résumé. La problématique d'extraction des itemsets fréquents dans un contexte de flot est ensuite rappelée. Puis, le fonctionnement général de l'approche est présenté. Enfin, les modifications mineures à apporter à la structure sont décrites.

5.1 Motivations

Un parallèle intéressant avec la mémoire humaine

Devant les similitudes existants entre le fonctionnement de la mémoire humaine et notre problématique, nous nous sommes intéressés à la théorie *mémoire constructive* [Mil56, SA07] qui considère que la mémoire humaine ne fonctionne pas par simple *accumulation* de souvenirs mais plutôt par *association* de souvenirs. D'une certaine manière, la sémantique des itemsets fréquents présente des similarités avec cette théorie. En effet, un itemset fréquent représente justement une association fréquente entre items (concepts). Proposer un résumé de itemsets fréquents d'un flot

offrirait des capacités d'analyse intéressantes pour un décideur qui disposerait d'un outil pour mesurer l'évolution des relations entre concepts.

Limites de l'extraction d'itemsets fréquents sur des batches

Nous revenons ici sur la discussion du chapitre 2. Une des deux méthodes existantes pour extraire les itemsets fréquents d'un flot est l'extraction d'itemsets fréquents sur des fenêtres temporelles du flot. La découverte des itemsets fréquents sur des sous-parties du flot, permet de mesurer l'évolution des comportements fréquents. L'inconvénient majeur de cette technique est que la taille du résumé produit augmente à chaque fois qu'un batch est fouillé. Dès lors, ce type de résumé ne respecte pas le critère de compacité. Il est donc nécessaire de proposer une structure pour unifier et synthétiser ces résultats en temps réel.

5.2 Extraction d'itemsets multidimensionnels fréquents dans des batches

Initialement introduite dans [AS94], la problématique de l'extraction d'itemsets fréquents dans une base de données \mathcal{D} consiste à rechercher les itemsets apparaissant dans au moins $minSupp\%$ des transactions de \mathcal{D} . L'adaptation de cette problématique au cas multidimensionnel peut être faite de la manière suivante. Soit $\mathcal{I} = \{x^1, x^2, \dots, x^m\}$ (avec $x^i = (x_1^i, \dots, x_N^i)$) un ensemble de m items N-multidimensionnels. Un k -itemset est un sous-ensemble de \mathcal{I} de cardinalité k . Le support d'un itemset X , noté $supp(X)$, est défini comme suit : $supp(X) = D_X/|\mathcal{D}|$ où D_X est le nombre de transactions dans lesquelles X apparaît et $|\mathcal{D}|$ est le nombre de transactions de la base \mathcal{D} . X est considéré comme fréquent si $supp(X) \geq minSupp$ où $minSupp$ est un paramètre numérique défini par l'utilisateur. Un itemset I est dit *maximal* [GZ02, BCF⁺05] s'il est fréquent et qu'il n'existe pas d'itemset fréquent inclus dans I .

Dans un contexte aussi dynamique que les flots de données, il n'est plus possible de considérer l'ensemble des transactions du flot pour calculer le support. Le support d'un itemset est alors calculé sur ces batches. Soit un flot $F = B_0, B_1, \dots$. Le support d'un itemset X dans un batch B_i est $supp_{B_i}(X) = D_X^i/|B_i|$ où D_X^i est le nombre de transactions de B_i dans lesquelles X apparaît et $|B_i|$ est le nombre de transactions de B_i .

5.3 Principe général

Un algorithme d'extraction d'itemsets fréquents sur des données statiques à une dimension [BCF⁺05, GZ02] est appliqué sur chaque batch. Pour cela, chaque item multidimensionnel dis-

Un batch est converti en un entier. Pour chaque batch B , nous obtenons un ensemble d'itemsets fréquents I_1, \dots, I_k et leur support associé sup_1, \dots, sup_k . Notons que les itemsets fréquents obtenus sont des itemsets de bas niveau dans le sens où ils ne sont composés que d'items multidimensionnels définis sur les plus fins niveaux de granularité. Un élément à insérer dans la structure est de la forme $E = (Id, val)$ tel que Id désigne un itemset fréquent dans le batch B_i et val le support de Id dans le batch concerné. La figure 4.3 illustre ce processus. Le changement de nature du résumé (*i.e.* des itemsets fréquents plutôt que des données brutes) implique de reconsidérer où seront stockés les différents historiques. Les modifications de la structure décrite dans la section précédente sont décrites ci-dessous.

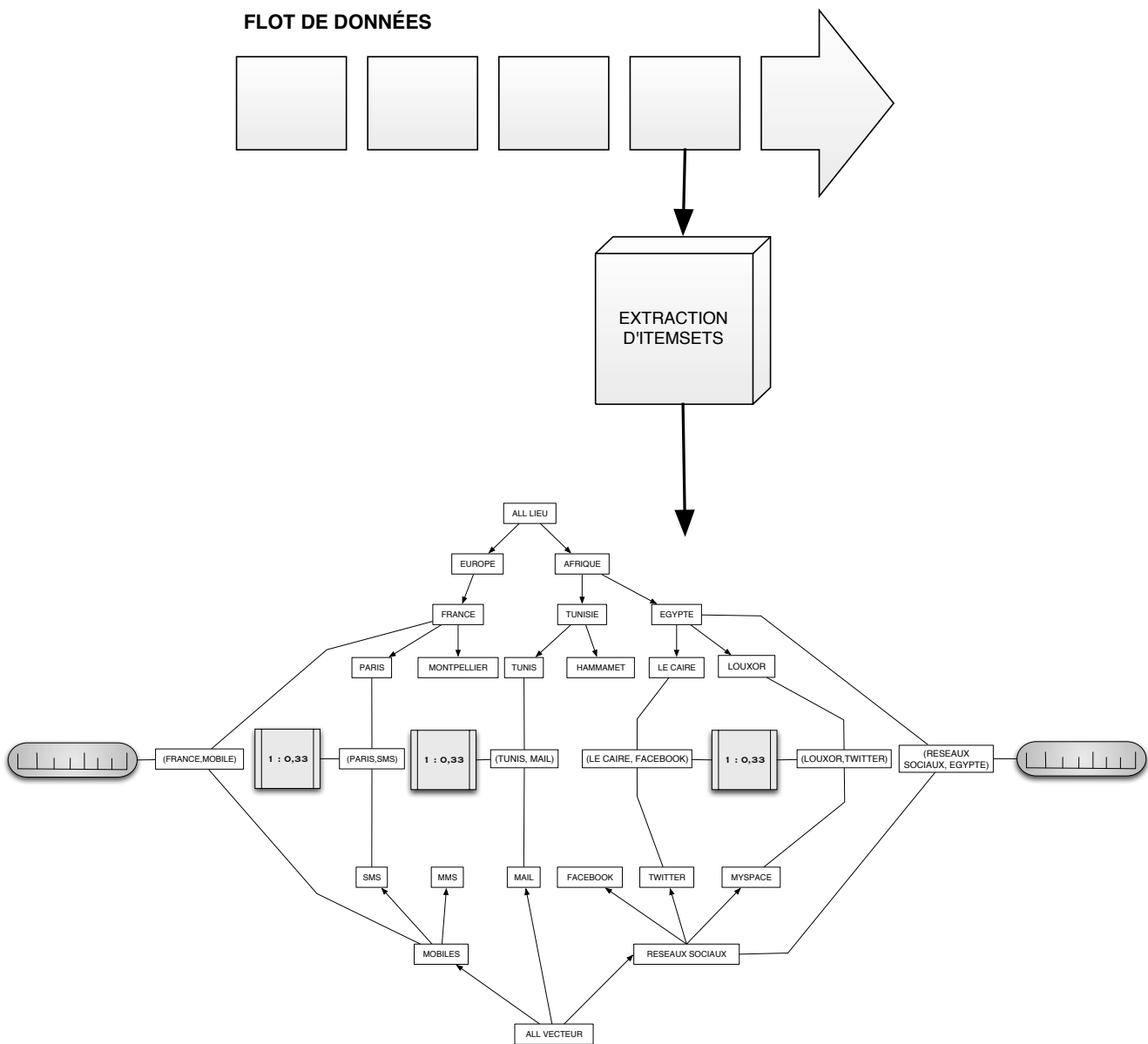


FIGURE 4.3 – Processus général pour la synthèse d'itemsets fréquents.

5.4 Adaptation de la structure existante

Les principales modifications à apporter à cette approche pour permettre la synthèse d'itemsets fréquents sont essentiellement structurelles et résident dans la détermination des nœuds qui stockeront l'historique des apparitions des itemsets.

Catégories de nœuds

Les nœuds structurels (NS) Cet ensemble est désormais composé des hiérarchies associées aux différentes dimensions D_1, \dots, D_n , des items multidimensionnels de bas niveau et des items multidimensionnels généralisés aux niveaux des cuboïdes du *popular path* $\mathbf{P} = C_1, \dots, C_k$.

Les nœuds de bas niveau (NBN) Un nœud de bas niveau stocke désormais l'historique des apparitions fréquentes d'un itemset de bas niveau grâce à une liste dynamique.

Les nœuds généralisés (NG) Ces nœuds conservent l'historique des généralisations des itemsets fréquents du flot grâce à un mécanisme de *tilted time window*.

Mise à jour

Les différents mécanismes de mise à jour et de propagation des données dans la structure décrits dans la section précédente sont facilement transposables pour permettre de synthétiser des itemsets fréquents extraits dans des batchs. Nous illustrons maintenant cette synthèse grâce au cas d'étude présenté dans le chapitre 1.

Exemple 4.2 *Nous supposons que l'algorithme d'extraction utilisé permet la découverte des itemsets fréquents maximaux et que $\min\text{Supp} = 0,2$. Sous ces conditions, l'ensemble \mathcal{I} des itemsets fréquents du batch 1 est $\mathcal{I} = \{((\text{Le Caire}, \text{Facebook})(\text{Louxor}, \text{Twitter})), ((\text{Tunis}, \text{Mail})), ((\text{Paris}, \text{SMS}))\}$. La figure 4.4 représente la structure de synthèse d'itemsets fréquents après l'insertion de ces itemsets. Nous remarquons que la nature des nœuds représentant les items multidimensionnels est différente par rapport au cadre de résumé des données brutes du flot. En effet, les items (e.g. $(\text{Le Caire}, \text{Facebook})$) sont désormais représentés par des nœuds structurels. Chaque nœud stockant l'historique des batchs où un itemset a été fréquent est relié aux nœuds structurels représentant les items le composant. Par exemple, le nœud associé à l'itemset $((\text{Le Caire}, \text{Facebook})(\text{Louxor}, \text{Twitter}))$ est relié aux nœuds associés aux items $(\text{Le Caire}, \text{Facebook})$ et $(\text{Louxor}, \text{Twitter})$. Nous remarquons que lorsqu'un itemset est composé d'un seul item, deux nœuds sont créés malgré tout (i.e. un nœud structurel pour représenter l'item*

et un nœud de bas niveau ou généralisé pour représenter l'itemset). Nous justifions ce choix par la différence de sémantique associée à ces deux nœuds.

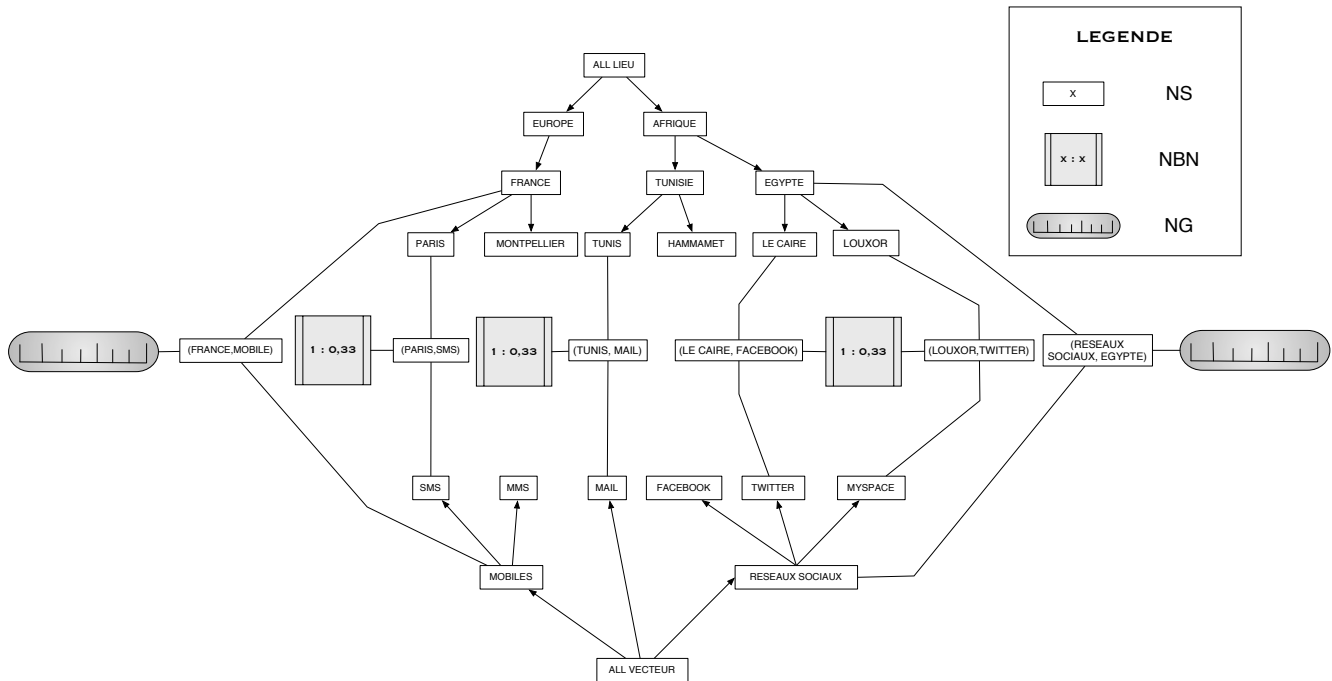


FIGURE 4.4 – Illustration de la structure proposée pour synthétiser des itemsets fréquents.

6 Expérimentations

Dans cette section, nous évaluons les performances de l'approche proposée dans ce chapitre pour en évaluer la faisabilité. Pour cela, nous procédons en deux temps. Premièrement, nous évaluons les critères de compacité et de rapidité de la mise à jour en étudiant l'impact des paramètres associés à cette proposition sur la consommation mémoire et le temps de mise à jour de la structure. Cette série de tests permet ainsi de valider expérimentalement le passage à l'échelle de l'approche. Ces expérimentations ont été menées aussi bien sur des jeux de données synthétiques que réelles et appliquées au résumé de données brutes tout comme à la synthèse d'itemsets fréquents. Nous mesurons ensuite la représentativité du résumé produit en reconstituant l'historique résumé des apparitions d'une cellule de bas niveau et en le comparant à l'historique réel. Les expérimentations ont été menées sur une machine équipée Intel(R) Xeon(R) CPU E5450 @ 3.00GHz avec 2Go de RAM et tournant sous Ubuntu 9.04. Les algorithmes ont été implémentés en Java 1.6.

6.1 Validation expérimentale du passage à l'échelle

Résumé de données brutes

Données synthétiques Les jeux de test synthétiques proviennent d'un générateur de batchs de données multidimensionnelles aléatoires satisfaisant certaines contraintes. La convention pour nommer un jeu de données est la suivante. $DdLlCcBbTtMmWw$ signifie qu'il y a d dimensions avec l niveaux (excepté le niveau ALL) chacune, que les arbres des hiérarchies sont de degré c , que les tests ont été réalisés sur $b \times 10^3$ batchs de t n-uplets et que les paramètres liés à l'approche sont $T_MAX = m$ et $W_{max} = w$. Nous présentons maintenant quelques résultats représentatifs obtenus pour mesurer l'impact des paramètres D, L et M . La *tilted time window* utilisée dans cette série d'expérimentations est $T = \langle W_1, \dots, W_{10} \rangle$ avec $Card(W_i) = 5$.

Influence du nombre de dimensions Nous évaluons d'abord l'influence du nombre de dimensions. Le jeu de données utilisé est $DxL3C5B20T100M10W5$ avec $x \in \{3, 10, 15\}$. La figure 4.5 présente quelques résultats obtenus. Puisque la répartition des données suit une loi de distribution uniforme, le nombre de dimensions a un impact significatif sur le nombre d'items multidimensionnels différents pouvant apparaître dans le flot de données. Cette remarque justifie l'augmentation de la taille de la structure (figures 4.5(b), 4.5(d) et 4.5(f)). Néanmoins, en observant ces trois courbes, on peut observer trois phases distinctes concernant la consommation mémoire. Dans un premier temps, la taille de la structure augmente très rapidement car, initialement, les nœuds de bas niveaux ne sont pas créés et sont donc construits au fur et à mesure que les items correspondant arrivent. Pendant la seconde phase, la taille de la structure augmente un peu plus lentement témoignant ainsi que deux phénomènes se produisent : la création de nouveaux nœuds de bas niveau et la fusion d'éléments dans les listes entraînant la propagation des informations le long des *tilted time windows* associées qu'il faut créer si elles n'existent pas. Enfin, la troisième phase est celle de la stabilisation de la taille de la structure. Une fois arrivée dans cette phase, tous les nœuds de bas niveau et généraux sont créés et il ne s'agit plus alors que de réaliser des opérations d'insertion d'éléments dans les listes, de fusion et de propagation dans les nœuds généralisés. Ces trois comportements sont aisément repérables sur la figure 4.5(b) à cause du nombre relativement faible d'items potentiels. Pour un nombre de dimensions plus important (figures 4.5(d) et 4.5(f)), cette troisième phase est plus longue à atteindre à cause du nombre important d'items différents dans le flot.

Concernant la durée d'insertion des données d'un batch dans la structure, (figures 4.5(a), 4.5(c) et 4.5(e)), nous remarquons que (1) la plupart des batchs sont traités en 50 ms, (2) il existe de rares batchs où le temps de traitement est anormalement élevé et peut être attribué au lancement

du ramasse miettes de la JVM et (3) le nombre de dimensions a finalement peu d'impact sur le temps de mise à jour de la structure. En outre, nous constatons la présence de trois catégories de batchs. La première correspond aux batchs où les éléments sont simplement ajoutés dans les listes, la seconde témoigne des mécanismes de fusion et de propagation et la troisième correspond aux batchs où il faut à la fois fusionner, propager et supprimer les éléments les plus anciens de la liste.

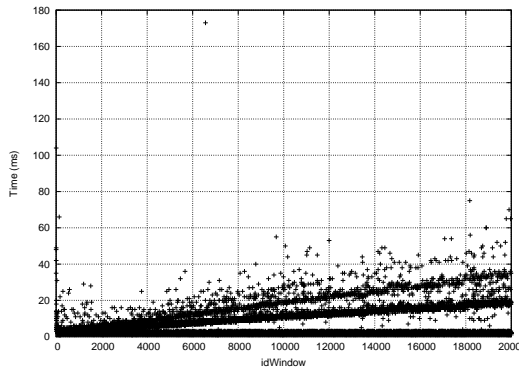
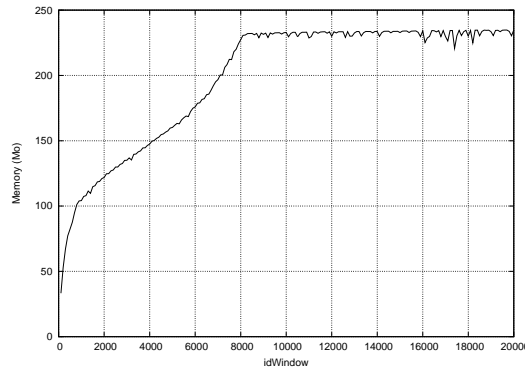
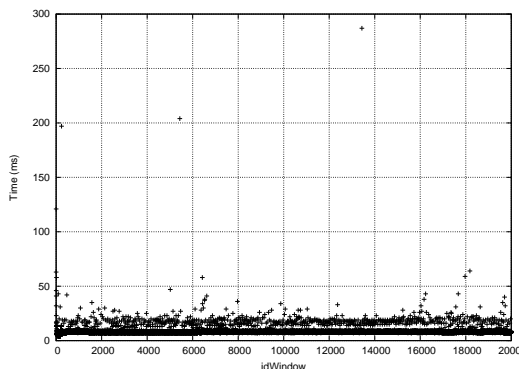
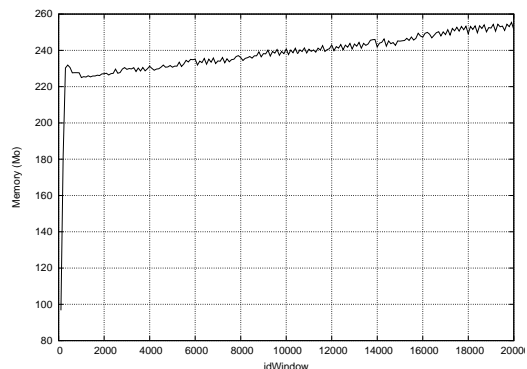
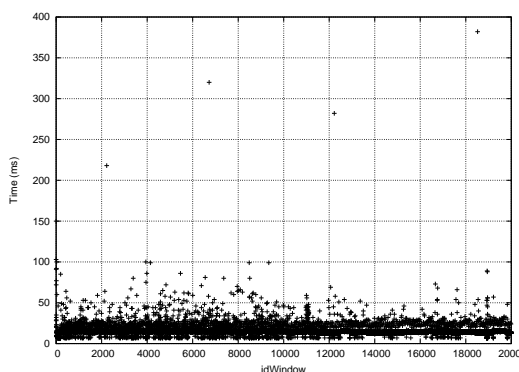
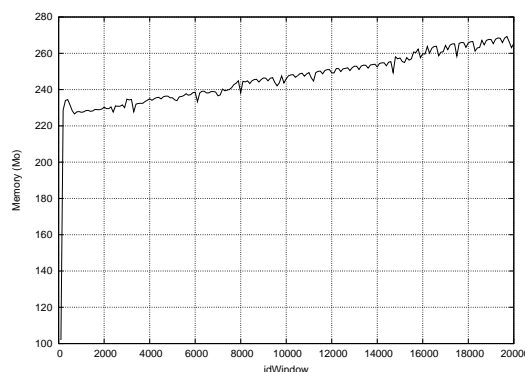
(a) Tps de mise à jour/batch ($x = 3$)(b) Consommation mémoire ($x = 3$)(c) Tps de mise à jour/batch ($x = 10$)(d) Consommation mémoire ($x = 10$)(e) ITps de mise à jour/batch ($x = 15$)(f) Consommation mémoire ($x = 15$)

FIGURE 4.5 – Influence du nombre de dimensions ($DxL3C5B20T100M10W5$)

Influence de la hauteur des hiérarchies Nous évaluons maintenant l'impact du nombre de niveaux dans les hiérarchies sur le temps d'insertion d'un batch dans la structure et la consommation mémoire. Le jeu de données utilisé est $D10LxC5B20T100SM10W5$ avec $x \in \{3, 7\}$. Concernant la taille de cette structure (figures 4.6(b) et 4.6(d)), ce paramètre influant davantage que le nombre de dimensions sur le nombre d'items différents potentiels, son impact est plus marqué. On retrouve malgré tout les deux premières phases décrites précédemment.

Concernant le temps de mise à jour de la structure (figures 4.6(a) et 4.6(c)), nous notons qu'ici aussi, l'augmentation du nombre d'items différents dans le flot impact réellement sur les résultats obtenus. En effet, à cause de cette plus grande disparité dans la distribution d'apparition des items, beaucoup moins de fusions sont réalisées et il est donc plus délicat de distinguer les trois catégories de batchs observées lors de la série d'expérimentations précédentes. On peut également remarquer que la courbe formée par les temps d'insertion anormalement élevés de certains batchs est très similaire à la courbe de la consommation mémoire. Ceci s'explique par le fait que le temps de traitement du ramasse miettes de la JVM est proportionnel à la taille de la structure à nettoyer.

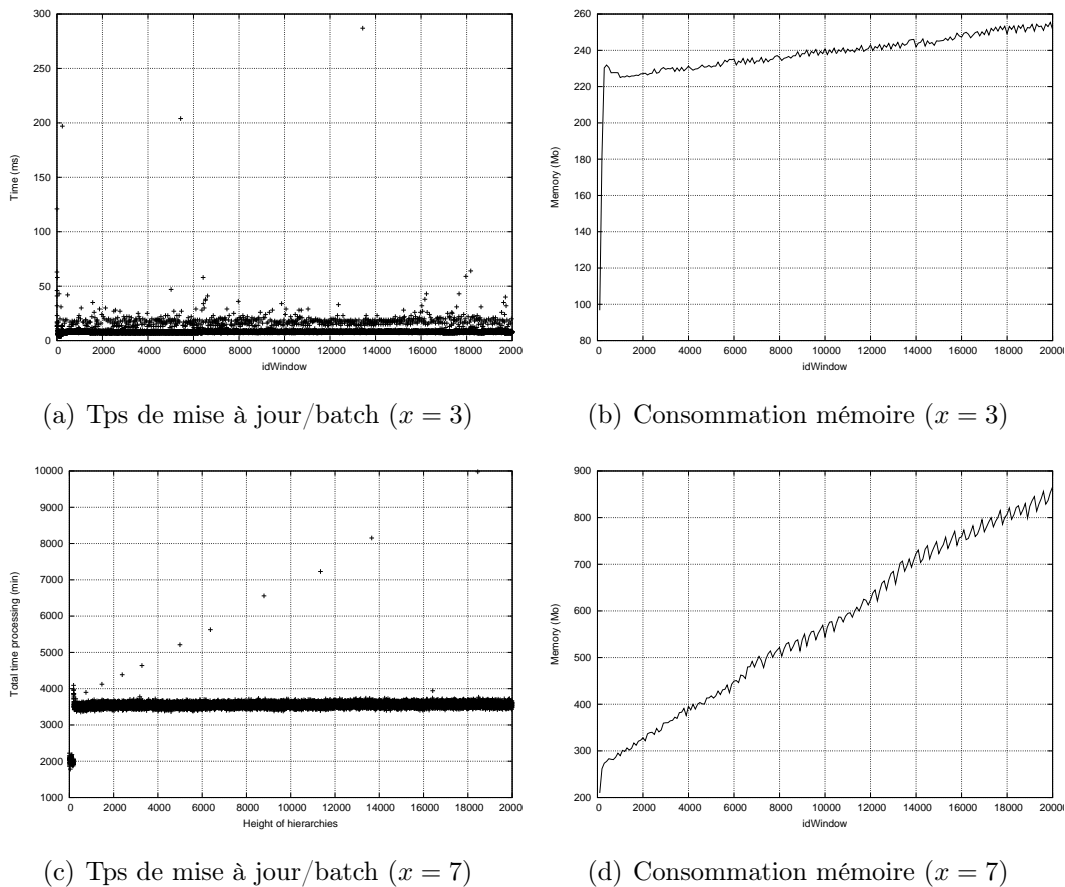


FIGURE 4.6 – Influence de la hauteur des hiérarchies ($D10LxC5B20T100SM10W5$)

Influence du paramètre MAX-SIZE Nous évaluons maintenant l'impact du nombre de niveaux dans les hiérarchies sur le temps d'insertion d'un batch dans la structure et la consommation mémoire. Le jeu de données utilisé est $D10L3C5B20T100Mx$ avec $x \in \{1, 10\}$. Concernant la taille de la structure (figures 4.7(b) et 4.7(d)), nous remarquons que ce paramètre n'influe finalement que très peu. Ceci peut s'expliquer par le fait que l'espace mémoire requis pour un nœud de bas niveau est principalement attribué par les *méta données* associées à ce nœud (*e.g.* identifiant de l'item représenté, lien avec les autres nœuds de la structure). Concernant le temps de mise à jour de la structure, nous notons qu'ici aussi, ce paramètre n'a que peu d'impact et les remarques faites lors des précédentes analyses restent valables.

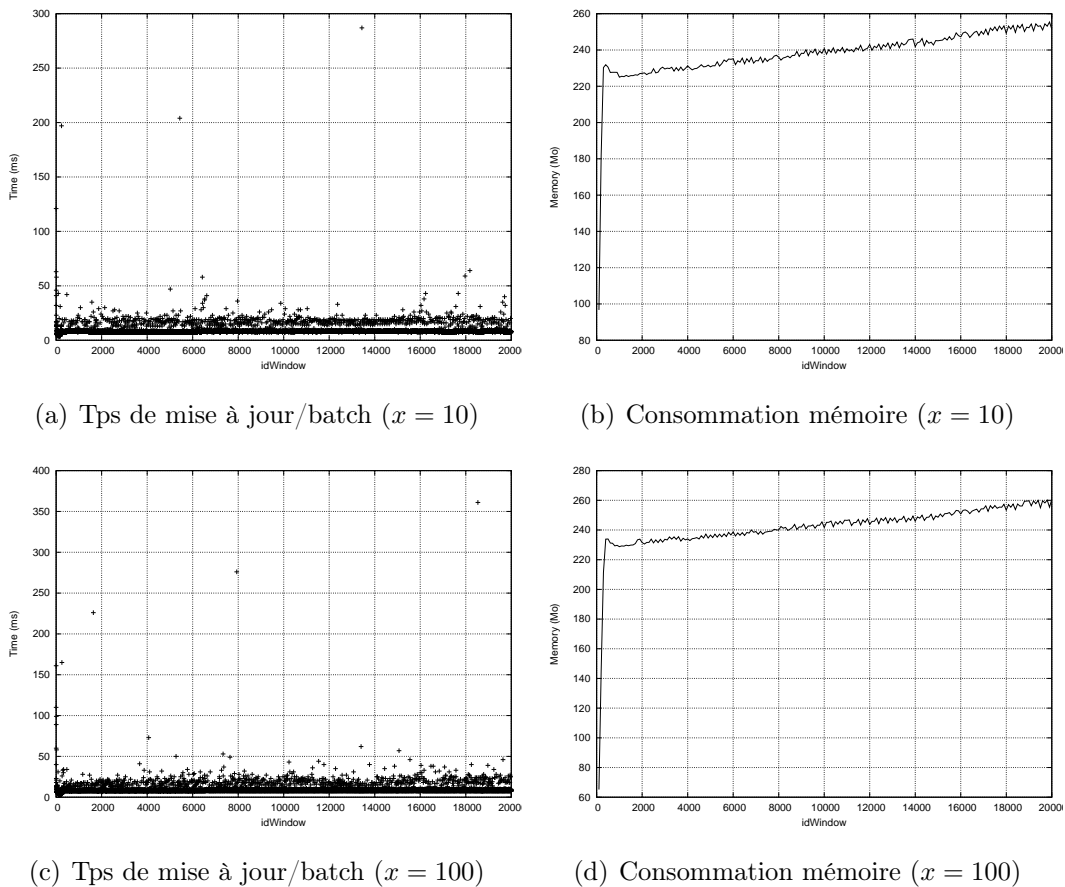


FIGURE 4.7 – Influence du paramètre T_MAX ($D10L3C5B20T100Mx$)

Ces expérimentations menées sur des données synthétiques montrent la robustesse de notre approche face à la diversité des données (nombre de dimensions, degré et profondeurs des hiérarchies, ...) dans une grande majorité de cas. Diversifier les données source engendre un coût de traitement plus important qui reste cependant acceptable.

Données réelles

Les données utilisées dans cette série d'expérimentations sont obtenues par des capteurs mesurant différents indicateurs physiques (*e.g.* humidité, température extérieure) sur des pompes industrielles. Un item d'un batch B est de la forme (d_1, \dots, d_{10}, m) où chaque d_i correspond à une valeur retournée par le capteur i dans le batch B et m représente le nombre d'apparition de (d_1, \dots, d_{10}) dans B . Sur chacune de ces dimensions, nous avons construit artificiellement des hiérarchies (*i.e.* une généralisation d'une valeur de bas niveau est un intervalle comprenant cette valeur) de hauteur 3 (excepté le niveau *ALL*) et, en moyenne, chaque nœud d'une hiérarchie possède 100 fils. Le jeu de données est dense dans le sens où les valeurs émises par chaque capteur sont relativement constantes. Le jeu de données a été découpé en batchs de 100 éléments. La *tilted time window* utilisée dans cette série d'expérimentations est $T = \langle W_1, \dots, W_{10} \rangle$ avec $Card(W_i) = 5$. Les valeurs des paramètres associés à l'approche sont $T_MAX = 100$ et $W_{max} = W_5$.

La figure 4.8 présente les résultats obtenus sur ce jeu de données. Si l'on considère l'évolution de la taille de la structure (figure 4.8(b)), nous notons que celle-ci devient rapidement stable (autour de 220Mo). Cela s'explique par la densité du jeu de données impliquant un nombre réduit de nœuds dans la structure. Si l'on s'intéresse maintenant au temps de mise à jour de la structure (figure 4.8(a)) on peut remarquer que (1) le temps moyen de mise à jour est d'environ 50ms, (2) que les différentes catégories de batchs décrites lors des expérimentations sur données synthétiques sont présentes, (3) le temps de mise à jour est relativement stable et (4), dans le pire des cas, ce délai est de 230 ms.

Ainsi, à travers cette série d'expérimentations, nous avons empiriquement montré que la résumé de données brutes d'un flot respecte les critères de compacité et de mise à jour rapide.

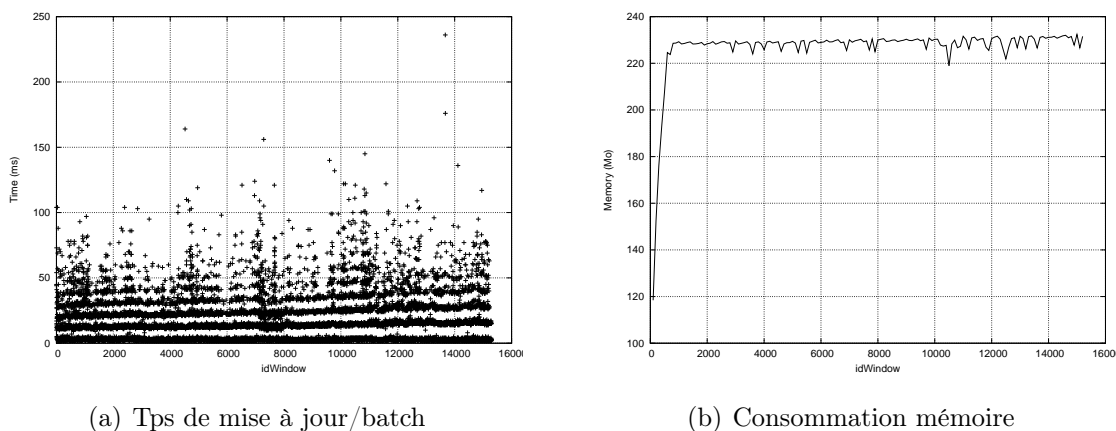


FIGURE 4.8 – Résumé de données brutes issues d'un jeu de données réelles

6.2 Synthèse d'itemsets fréquents

Vu la proximité des résultats obtenus pour la synthèse d'itemsets fréquents par rapport à ceux obtenus pour le résumé de données brutes, nous ne reportons ici que les résultats obtenus sur les données réelles précédemment décrites.

La méthode utilisée pour extraire les itemsets de bas niveau fréquents est la suivante. Comme précédemment, le jeu de données est découpé en batchs de 1000 éléments chacun. Un élément est de la forme (id, d_1, \dots, d_{10}) où $id \in \{1, \dots, 10\}$ désigne l'identifiant d'un client et chaque d_i correspond à une valeur retournée par le capteur i . Dans la mesure où nous ne disposons pas d'une implémentation d'un algorithme d'extraction d'itemsets multidimensionnels, nous transformons chaque (d_1, \dots, d_{10}) distinct en un entier. L'algorithme FP-GROWTH³ [JJY00] est ensuite exécuté sur chaque batch avec un support minimum de 10%. Il est important de noter que les résultats retranscrits ici ne tiennent pas compte de cette étape d'extraction.

La figure 4.9(b) présente l'évolution de la taille de la structure. Nous remarquons que celle-ci se stabilise rapidement et cela s'explique par la proximité des itemsets fréquents dans les différents batchs (*i.e.* un nombre limité de nœuds a donc été créé). Si nous nous intéressons maintenant au temps de mise à jour de la structure présenté dans la figure 4.9(a), celui-ci est généralement inférieur à 20 ms. Lorsque celui-ci est supérieur, cela témoigne d'opérations de fusion et de propagation. Notons que ce temps est légèrement supérieur au temps de fusion et propagation dans un contexte de résumé de données brutes à cause de la plus grande complexité des objets résumés (*i.e.* il est plus long de calculer la généralisation d'un itemset que d'un item).

3. l'implémentation utilisée est celle disponible sur le site du projet IlliMine (<http://illimine.cs.uiuc.edu/>)

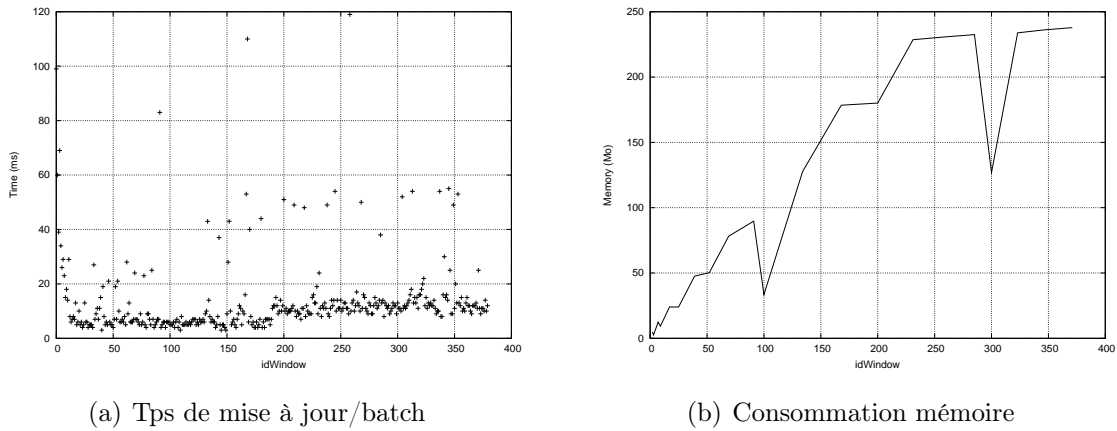


FIGURE 4.9 – Synthèse d'itemsets issus de données réelles

6.3 Représentativité du résumé produit

La représentativité du résumé produit est évaluée ici sur le jeu de données réelles décrit ci-dessus. Pour cela, nous adoptons la méthodologie suivante. Le résumé est construit sur les données brutes du flot. Les valeurs des paramètres associés à l'approche sont $T_MAX = 100$ et $W_{max} = W_5$. Le flot a été découpé en batchs de 100 n-uplets chacun. La fonction d'agrégation choisie pour la fusion des éléments dans une liste est la moyenne. La *titled time window* utilisée est identique à celle utilisée dans l'expérimentation précédente sur données réelles. Le résumé de quelques capteurs a été reconstruit une fois le jeu de données résumé. Pour cela, nous avons parcouru les nœuds de bas niveau et généralisés. Nous choisissons de présenter le résultat de cette méthode sur deux types de capteurs : un dont les valeurs sont plutôt changeantes et un dont les valeurs sont plutôt stationnaires

La figure 4.10 présente la comparaison entre l'historique réel (figure 4.10(a)) et le résumé (figure 4.10(b)) d'un capteur aux valeurs plutôt changeantes. Même si la valeur du paramètre T_MAX est faible, le résumé produit représente assez fidèlement les données réellement émises malgré l'utilisation de la moyenne dans le mécanisme de fusion qui nivelle légèrement les évolutions. Malgré tout, les différentes plages de valeurs apparaissent clairement dans le résumé produit.

La figure 4.11 présente la comparaison entre l'historique réel (figure 4.11(a)) et le résumé (figure 4.11(b)) d'un capteur aux valeurs plutôt stables. Pour ce genre de capteur, le résumé retranscrit très fidèlement l'historique des valeurs émises.

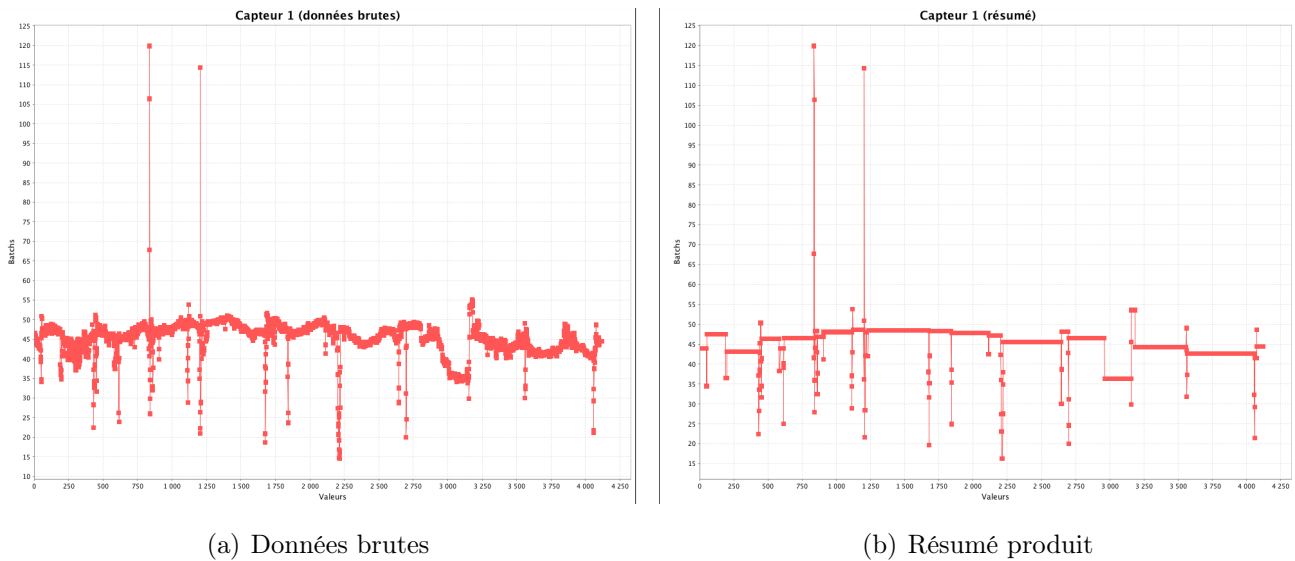


FIGURE 4.10 – Données brutes vs. résumé produit (valeurs changeantes)

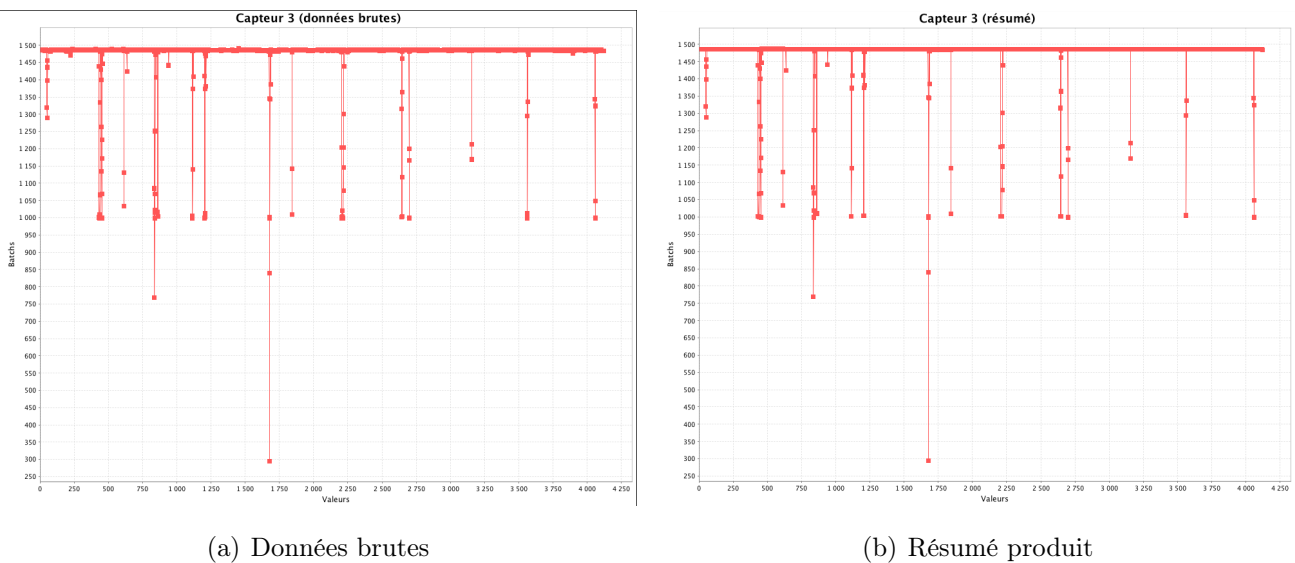


FIGURE 4.11 – Données brutes vs. résumé produit (valeurs stables)

7 Discussion

Dans ce chapitre, nous définissons une structure unique pour résumer aussi bien des données brutes du flot que des itemsets fréquents. Nous proposons des mécanismes pour la mettre à jour dont l'efficacité est validée par une série d'expérimentations. En proposant un mécanisme de listes dynamiques pour stocker l'historique des apparitions des éléments de bas niveau, nous pallions une limite des *titled time windows* et améliorons ainsi la représentativité du résumé produit. Une

série d'expérimentations menées à ce sujet confirme cette affirmation.

Les perspectives associées à ce travail sont nombreuses. Nous en citons ici quelques unes dont certaines seront reprises dans la discussion finale de ce manuscrit.

Dans notre proposition, nous supposons que les hiérarchies sont des arbres. En pratique, les hiérarchies sont généralement des graphes orientés sans cycle (DAG) et un même item peut posséder plusieurs généralisations (*i.e.* il peut exister plusieurs chemins possible entre la racine de la hiérarchie et un nœud du graphe). Toutefois, de part la structure de graphe utilisée dans cette proposition, il serait aisé d'intégrer ce type de hiérarchies.

Concernant le résumé d'itemset fréquents, l'utilisation d'un algorithme d'extraction n'intégrant ni l'aspect multidimensionnel des données ni les hiérarchies peut sembler limitative quant à la qualité du résumé produit. Dès lors, l'utilisation d'un algorithme d'extraction d'itemsets multidimensionnels et multiniveaux tel que celui proposé dans [SBW08] est une alternative possible. La synthèse de ces itemsets demanderait quelques ajustements quant à la définition des nœuds de bas niveau mais semble envisageable grâce à la structure de graphe adoptée.

L'utilisation de technique de fouille de données et la synthèse des résultats produits apparaît être une bonne technique de résumé car elle capture les tendances présentes dans un flot de données. Dès lors, il est naturel de se demander si des techniques de fouilles de données multidimensionnelles et multiniveaux autres que les itemsets fréquents peuvent être utilisées dans un tel contexte. Le prochain chapitre apporte une réponse affirmative à cette interrogation. Pour cela, nous proposons d'abord un algorithme d'extraction de motifs séquentiels multidimensionnels et multiniveaux permettant la découverte de motifs ne pouvant être obtenu par les approches existantes. Ensuite, nous introduisons une solution pour résumer un flot grâce à de tels motifs.

Chapitre 5

Des séquences fréquentes pour résumer un flot de données

1 Introduction

Les approches présentées dans les chapitres précédents se sont principalement focalisées sur la construction de résumés qui considéraient l'intégralité des données circulant sur le flot. Comme nous l'avons discuté dans le chapitre 2 et esquissé dans le chapitre précédent, les techniques de fouille de données peuvent également se révéler utiles et pertinentes dans notre contexte. Notamment, devant la séquentialité naturelle des données d'un flot, l'extraction des séquences fréquentes apparaît particulièrement adaptée pour caractériser les évolutions des tendances.

En particulier, nous nous focalisons sur l'exploitation des caractéristiques multidimensionnelles et multiniveaux dans ce processus d'extraction que peu d'approches exploitent aussi bien dans un contexte statique que dynamique. Cela s'explique principalement par l'espace de recherche immense à parcourir pour exhiber des séquences multidimensionnelles et multiniveaux. Dans ce chapitre, nous étudions l'intégration de ces spécificités dans les processus de fouille et de résumé et proposons pour cela deux approches, *SALINES* et *ALIZES*, utilisables selon la nature des données (*i.e.* statiques ou dynamiques).

Tout d'abord, nous proposons *SALINES*, un algorithme d'extraction de séquences multidimensionnelles et multiniveaux sur des bases de données statiques. L'originalité de cette approche est de permettre la découverte de séquences ne pouvant être extraites par les approches existantes. Pour cela, *SALINES* procède en deux temps. D'abord, un automate indexant les sous-séquences de la base de données à fouiller est construit. Cet automate est ensuite fouillé afin de découvrir des séquences multiniveaux. Cette seconde étape est réalisée grâce à un parcours en profondeur

de l'automate généré ainsi qu'à des mécanismes de fusions d'états. Les résultats obtenus sont satisfaisants tant sur le plan performance (*i.e.* consommation mémoire et temps d'extraction) que sur la qualité des motifs extraits (*i.e.* longueur des motifs et présence d'items généralisés).

Malgré les bonnes performances obtenues, *SALINES* procède en deux étapes et ne peut donc être appliqué directement sur un flot de données. Face à cette inadéquation, nous proposons *ALIZES*, une approche pour résumer un flot via des séquences multidimensionnelles et multi-niveaux. Dans *ALIZES*, les séquences fréquentes d'items de bas niveau sont extraites *en ligne* sur chaque batch. Nous utilisons pour cela *SPAMS* [VSMP09], un algorithme d'extraction de séquences fréquentes sur des flots générant un automate indexant ces séquences fréquentes par batch. Ces automates sont ensuite insérés dans une *titled time window*. L'agrégation d'automates pour le passage à une granularité temporelle est réalisée *hors ligne* grâce à l'union des automates et au parcours de l'automate fusionné pour exhiber des séquences multiniveaux. Nous montrons comment paralléliser ce processus pour améliorer les performances. Les résultats obtenus (consommation mémoire et temps de traitement) confirment les bonnes propriétés de notre approche.

Le plan de ce chapitre est le suivant. Dans la section 2, nous introduisons quelques notations, rappelons la problématique d'extraction des séquences fréquentes multidimensionnelles et multi-niveaux et présentons la structure d'automate que nous manipulons dans les deux approches proposées dans ce chapitre. La section 3 présente *SALINES*, une méthode d'extraction de séquences fréquentes inédites. L'approche *ALIZES* est décrite dans la section 4. Enfin, nous concluons ce chapitre en donnant quelques perspectives associées à ces deux approches. Certaines d'entre elles seront développées dans le chapitre final de ce manuscrit.

2 Concepts préliminaires et notations

2.1 Problématique de l'extraction des motifs séquentiels multidimensionnels

Nous nous attachons dans cette section à définir formellement les concepts d'items multidimensionnels et de séquences multidimensionnelles.

Définition 5.1 (*Item multidimensionnel*) *Un item multidimensionnel* $a = (d_1, \dots, d_M)$ *est un* n -*uplet tel que* $\forall i = 1, \dots, M$ *on a* $d_i \in \text{Dom}(D_i)$ *et qu'il existe au moins un* i *tel que* $d_i \neq \text{ALL}_i$.

Par exemple, $(Tunis, Twitter)$ et $(Europe, Mobiles)$ sont des items multidimensionnels. Notons que $(ALL_{Lieu}, ALL_{Vecteur})$ n'est pas considéré comme un item multidimensionnel. Cela est dû au fait qu'il représente une connaissance trop triviale. En ce sens, il n'apporte aucune information à l'utilisateur. Il est important de souligner qu'un item multidimensionnel peut être défini sur n'importe quel niveau des hiérarchies associées aux dimensions d'analyse. Cela constitue une différence fondamentale avec les éléments présents dans la base de séquences SDB qui eux ne sont décrits que sur les niveaux les plus précis des hiérarchies associées aux dimensions d'analyse. Dans la mesure où un item peut être considéré à n'importe quel niveau de précision, il est possible de comparer deux itemsets selon leur niveau de précision.

Définition 5.2 (*Relation entre items*) Soient $a = (d_1, \dots, d_M)$ et $a' = (d'_1, \dots, d'_M)$ deux itemsets multidimensionnels.

- a est dit plus général que a' (on notera $a' \subseteq a$) si $\forall i = 1, \dots, M$, d_i est un ancêtre de d'_i dans H_i ou $d_i = d'_i$. On notera $a' \subset a$ si au moins un d_i est un ancêtre de d'_i .
- a est dit plus spécifique que a' (on notera $a \subseteq a'$) si $\forall i = 1, \dots, M$, d'_i est un ancêtre de d_i dans H_i ou $d'_i = d_i$. On notera $a \subset a'$ si au moins un d'_i est un ancêtre de d_i .
- a et a' sont incomparables s'il n'existe aucune relation entre a et a' ($a' \not\subseteq a$ et $a \not\subseteq a'$)

Par exemple, l'item $(France, Facebook)$ est plus général que l'item $(Paris, Facebook)$. Par contre, les items $(France, Facebook)$ et $(Paris, Réseaux sociaux)$ sont incomparables.

Un item $a = (d_1, \dots, d_M)$ est de bas niveau si $\forall i = 1, \dots, M$, d_i est une feuille de H_i . Maintenant que nous avons défini la notion d'item multidimensionnel, nous pouvons définir une séquence multidimensionnelle.

Définition 5.3 (*Séquence multidimensionnelle*) Une k -séquence multidimensionnelle $s = \langle e_1, \dots, e_k \rangle$ est une suite ordonnée de k items multidimensionnels.

Exemple 5.1 Un exemple de séquence multidimensionnelle est $\langle (Montpellier, Tweeter)(Tunis, SMS) \rangle$.

Une séquence multidimensionnelle est de bas niveau si tous les items qui la composent sont de bas niveau. On dit qu'une séquence (multidimensionnelle) S supporte une séquence $s = \langle e_1, \dots, e_k \rangle$ si $\forall e_i \in s$, il existe dans S un item e'_i tel que $e' \subseteq e$ et que la relation d'ordre est respectée. Si l'on considère la base de séquences présentée dans le tableau 5.1, nous avons S_1 qui supporte la

séquence $\langle (Tunis, Facebook)(Tunisie, Mail) \rangle$.

Le support d'une séquence s est le nombre de séquences de SDB qui supportent s . Soit σ un paramètre numérique défini par l'utilisateur. On dit qu'une séquence s est fréquente si son support est supérieur ou égal à σ .

La problématique de l'extraction de motifs séquentiels multidimensionnels peut être définie comme la recherche dans une base de séquences multidimensionnelles de toutes les séquences dont le support est supérieur ou égal au paramètre utilisateur σ .

<i>Clients</i>	Séquences de données
2	$\langle (Tunis, Facebook)(Tunis, Facebook)(Hammamet, Mail) \rangle$
3	$\langle (Paris, Facebook)(Tunis, Tweeter)(Hammamet, Mail) \rangle$

TABLE 5.1 – La base SDB considérée dans ce chapitre (*i.e.* le batch 3 du cas d'étude restreint sur la dimension *Vecteur de communication*)

2.2 Un automate indexant les séquences

Les approches développées dans ce chapitre exploitent une structure d'automate pour indexer les séquences et les motifs fréquents. Nous consacrons cette section à la description de cette structure et aux notations utilisées pour la manipuler. Pour illustrer ces définitions, nous nous appuyons sur la figure 5.1 qui présente l'automate indexant toutes les sous-séquences issues du troisième batch de notre cas d'étude. Par souci de clarté, nous ne considérons que les membres de la dimension *Vecteur de communication*.

L'automate manipulé est un automate fini déterministe pouvant être défini par un quintuplet $\mathcal{A} = (\mathcal{Q}, q_0, F, \mathbb{I}, \delta)$ où :

- \mathcal{Q} est un ensemble fini d'états dont chaque état est représenté par un identifiant unique ;
- $q_0 \in \mathcal{Q}$ est l'état initial ;
- $F \subseteq \mathcal{Q}$ est l'ensemble des états finaux ;
- $\Sigma \subseteq \mathbb{I}$ est l'alphabet des items reconnus ;
- $\delta : \mathcal{Q} \times \Sigma \rightarrow \mathcal{Q}$ est la fonction de transition non totale permettant d'indexer les motifs séquentiels fréquents.

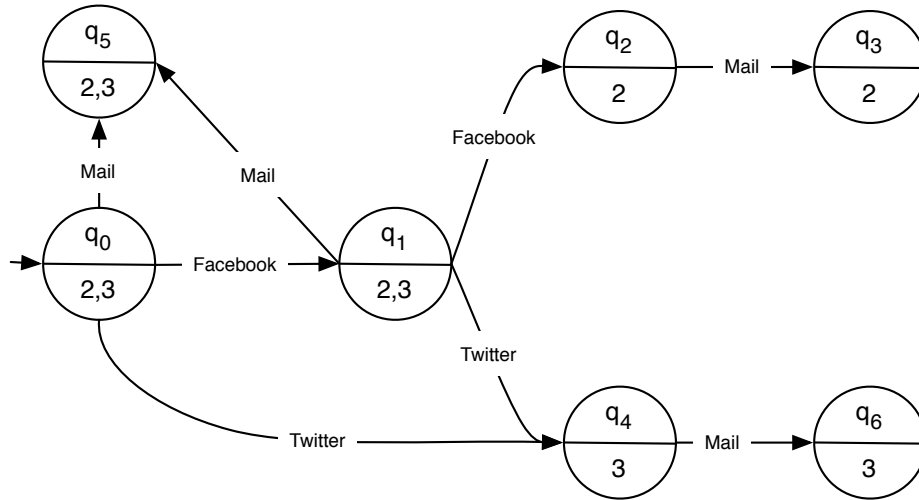


FIGURE 5.1 – Automate indexant les sous-séquences des données du troisième batch restreintes sur la dimension *Vecteur de communication* ($\sigma = 1$)

Chaque transition est labellisée par un item et les transitions entrantes d'un état sont labellisées par le même item. Puisque l'automate est déterministe, il ne peut y avoir deux transitions sortantes possédant un label identique pour chaque état. Une séquence s est indexée par un état q s'il existe un chemin de l'état initial vers cet état q tel que les transitions de ce chemin représentent s . Par exemple, la séquence $\langle\langle Facebook \rangle\rangle(Twitter)\rangle$ est reconnue par l'état q_3 . Un ensemble de clients est associé à chaque état et indique quels sont les clients qui supportent les séquences indexées par un état. Par exemple, les clients 2 et 3 supportent tous deux les séquences indexées par un état q_3 (*i.e.* $\langle\langle Facebook \rangle\rangle(Tweeter)\rangle$ et $\langle\langle Tweeter \rangle\rangle$). De plus, il ne peut exister un chemin entre deux états q et q' que si l'ensemble des clients associés à q' est un sous-ensemble des clients associés à q . Un état est dit *final* si les séquences indexées par cet état sont fréquentes (au sens du support minimum fixé par l'utilisateur).

Maintenant que nous avons présenté les caractéristiques de cet automate, nous introduisons quelques notations et fonctions permettant sa manipulation. Ces notations sont résumées dans le tableau 5.2. Nous notons \mathcal{C} l'ensemble des clients présents dans la base de données. Dans l'exemple considéré ici, $\mathcal{C} = \{2, 3\}$. Pour un client C_i donné, nous définissons la fonction $States(C_i)$ qui retourne l'ensemble des états de l'automate tels que C_i est associé à cet état (*e.g.* $States(C_i) \subseteq \{q_0, q_1, q_2, q_4, q_6, q_7\}$). La fonction duale, $Customers(q)$ retourne, pour un état donné, l'ensemble des clients associés à cet état (*e.g.* $Customers(q_3) = \{2, 3\}$). La fonction $Trans_{in}(q)$ (resp. $Trans_{out}(q)$) permet de connaître pour un état q donné la liste des transitions entrantes (resp. sortantes) de q . Par exemple, $Trans_{in}(q_3) = \{q_0 \xrightarrow{Twitter} q_3, q_1 \xrightarrow{Twitter} q_3\}$. La fonction $Trans(q, e)$ permet de filtrer le contenu de $Trans_{out}(q)$ de la manière suivante. $Trans(q, e)$ retourne, pour un

\mathcal{C}	Ensemble des identifiants client
$States(C_i)$	Ensemble d'états pour un client C_i donné
$Customers(q)$	Ensemble des clients pour un état q donné
$Trans_{out}(q)$	Ensemble des transitions sortantes d'un état q donné
$Trans_{in}(q)$	Ensemble des transitions entrantes d'un état q donné
$Succ(q)$	Ensemble des successeurs d'un état q donné
$Succ(q, e)$	Ensemble des successeurs d'un état q donné accessibles par des transitions étiquetées e_i où e_i est un item de <i>bas niveau</i> et $e_i \subseteq e$
$Trans(q, e)$	Ensemble des transitions $q \xrightarrow{e_i} q'$ d'un état q donné tel que $q' \in Succ(q, e)$
$Pred(q)$	Ensemble des états précédant un état q donné
$SubPred(q) \subseteq Pred(q)$	Sous-ensemble des états précédant un état q donné
$SubPred$	Ensemble des états q tel qu'il existe $SubPred(q)$
$Seq(q)$	Ensemble des séquences reconnues par l'état q
\mathcal{T}	Ensemble des transitions de l'automate
q_∞	Etat puits (état non accessible et fictif)

TABLE 5.2 – Notations utilisées dans ce chapitre

état q et un item e donnés, l'ensemble des transitions sortantes de q labellisées par un item plus spécifique que e . Par exemple, $Trans(q_1, \text{Réseaux sociaux}) = \{q_1 \xrightarrow{\text{Twitter}} q_3, q_1 \xrightarrow{\text{Facebook}} q_2\}$. Similairement, nous introduisons la fonction $Pred(q)$ (resp. $Succ(q)$) qui retourne, pour un état q donné, l'ensemble des états prédécesseurs (resp. successeurs) de q . Un état q' précède (resp. succède) un état q s'il existe une transition $q' \xrightarrow{e_i} q$ (resp. $q \xrightarrow{e_i} q'$). Nous introduisons également une fonction $Succ(q, e)$ pour filtrer les états successeurs de q . $Succ(q, e)$ retourne ainsi la liste des états accessibles grâce aux transitions de $Trans(q, e)$ (e.g. $Succ(q_1, \text{Réseaux sociaux}) = \{q_2, q_3\}$). La fonction $Seq(q)$ permet de connaître toutes les séquences indexées par l'état q . Enfin, nous introduisons q_∞ , un état fictif utilisé dans l'étape de construction de l'automate des sous-séquences.

Ces notations introduites, nous présentons maintenant *SALINES*, un algorithme de fouille de motifs séquentiels multidimensionnels multiniveaux.

3 SALINES : un algorithme d'extraction de motifs multiniveaux

3.1 Aperçu général de l'approche

L'objectif de l'approche *SALINES* est de permettre la découverte de motifs séquentiels multidimensionnels multiniveaux à partir d'une base de séquence *SDB*. Pour cela, nous procédons en deux temps. L'automate des sous-séquences de *SDB* est d'abord construit. Il est ensuite parcouru pour en extraire l'ensemble des motifs multiniveaux fréquents. Pour réaliser la première étape, nous nous inspirons de l'algorithme SPAMS [VSMP09] qui permet la construction incrémentale d'un automate afin d'indexer les motifs séquentiels de bas niveaux sur un flot de données. Cette étape est détaillée dans la section 3.3. Notons que l'automate des sous-séquences initial n'indexe que des séquences fréquentes de bas niveau. Les séquences plus générales sont découvertes ensuite, lors d'une seconde étape. Au cours de celle-ci, un parcours en profondeur est réalisé pour rechercher des états à fusionner permettant de reconnaître des séquences fréquentes plus générales. La section 3.4 expose précisément ce mécanisme de fusion d'états. Pour simplifier l'exposé, nous présentons notre approche dans un contexte monodimensionnel hiérarchisé. Nous discutons dans la section 3.5 des modifications mineures à apporter pour gérer des données multidimensionnelles. Avant de décrire en détail notre approche, nous dressons un panorama des approches existantes dans la littérature relatives à l'extraction de motifs séquentiels multidimensionnels et multiniveaux dans la section 3.2.

3.2 Etat de l'art des algorithmes d'extraction de motifs multidimensionnels multiniveaux

A notre connaissance, il n'existe que deux méthodes qui exploitent les hiérarchies dans ce contexte [PLT06, PLT08] que nous détaillons maintenant.

L'algorithme HYPE ([PLT06]) est la première approche à aborder cette difficulté mais présente une faiblesse de taille : il n'est pas possible d'extraire des séquences contenant deux items *comparables*. Deux items multidimensionnels sont *comparables* si l'un est la généralisation de l'autre. Par exemple, l'item *(Paris, Tweeter)* est comparable à l'item *(France, Réseaux sociaux)*. Les auteurs proposent de travailler avec les items multidimensionnels les plus spécifiques. On appelle "*item fréquent le plus spécifique*" un item fréquent tel qu'il n'existe pas d'item plus spécifique fréquent. Par exemple si *(Paris, Tweeter)* est fréquent alors *(France, Tweeter)* l'est aussi. Seul *(Paris, Tweeter)* sera conservé. Comme tous les items multidimensionnels et séquences multidimensionnelles ne sont générées qu'à partir de ces items multidimensionnels les plus spécifiques, il est impossible de voir

apparaître deux items comparables (e.g. $(Paris, Tweeter)$ et $(France, Tweeter)$) dans une même séquence. Or, il se peut que cela empêche de trouver des séquences intéressantes. Par exemple si la séquence $\langle (France, Tweeter)(Tunisie, SMS) \rangle$ est fréquente, elle ne sera jamais retrouvée car seules les séquences contenant $(Paris, Tweeter)$ seront examinées.

Dans [PLT08], les auteurs définissent alors les concepts de séquence *convergente* et *divergente* et proposent un algorithme pour les extraire dans une base de données multidimensionnelles. Une séquence convergente est une séquence qui part d'une connaissance générale pour aller vers une connaissance plus précise. Par exemple, la séquence "quand la fréquence d'envoi de SMS à Lyon augmente, le nombre de messages envoyés depuis un mobile augmente ensuite en France" est une séquence convergente. A l'inverse, une séquence divergente est une séquence qui débute par une connaissance précise et qui se généralise par la suite. Cette approche ne permet pas d'extraire des séquences où il n'existe pas de conditions sur le niveau de ses items (i.e. où une séquence fréquente comporte des sous-séquences divergentes et d'autres convergentes).

Nous proposons une méthode originale, SALINES, qui exploite plus efficacement les hiérarchies et permet la découverte des séquences d'items multidimensionnels inédites. Nous détaillons maintenant son fonctionnement.

3.3 Phase 1 : construction de l'automate des sous-séquences

Algorithme 5.1: `main()`

```

1 Création des états  $q_0, q_\infty : \mathcal{Q} \leftarrow \{ q_0, q_\infty \}$ 
2  $Customers(q_0) \leftarrow \emptyset$ 
3  $Customers(q_\infty) \leftarrow \emptyset$ 
4  $\mathcal{T} \leftarrow \emptyset$ 
5 pour  $i \leftarrow 0$  à Nb éléments dans  $S_{SDB}$  faire
6    $\left[ \text{insert}(S_i, e_i) \right.$ 

```

Une fois la structure initialisée (algorithme 5.1), la construction de l'automate des sous-séquences est principalement assurée par la fonction *Insert* qui permet l'insertion de chaque item de *SDB*. Cette fonction permet de garantir que les caractéristiques de l'automate présentées dans la section précédente sont effectives.

La fonction *Insert* accepte deux paramètres en entrée : C_i désigne l'identifiant du client et e_i représente l'item à insérer. La première étape de l'algorithme consiste à vérifier si le client C_i est

Algorithme 5.2: insert(C_i, e_i)

```

1 si  $C_i \notin \mathcal{C}$  alors
2    $\mathcal{C} \leftarrow \mathcal{C} \cup \{ S_i \}$ 
3    $Customers(q_0) \leftarrow Customers(q_0) \cup \{ C_i \}$ 
4    $States(S_i) \leftarrow \{ q_0 \}$ 
5 pour chaque état  $q \in States(C_i)$  faire
6   si  $\nexists q' \in \mathcal{Q} \mid q \xrightarrow{e_i} q' \in \mathcal{T}$  alors
7      $SubPred(q_\infty) \leftarrow SubPred(q_\infty) \cup \{ q \}$ 
8      $SubPred \leftarrow SubPred \cup \{ q_\infty \}$ 
9   sinon
10     $SubPred(q') \leftarrow SubPred(q') \cup \{ q \}$ 
11     $SubPred \leftarrow SubPred \cup \{ q' \}$ 
12 pour chaque état  $q' \in SubPred$  faire
13   si  $q' \neq q_\infty$  and  $|Pred(q')| = |SubPred(q')|$  alors
14      $States(S_i) \leftarrow States(S_i) \cup \{ q' \}$ 
15      $Customers(q') \leftarrow Customers(q') \cup \{ S_i \}$ 
16   sinon
17     • Création de l'état  $q'' : \mathcal{Q} \leftarrow \mathcal{Q} \cup \{ q'' \}$ 
18      $States(S_i) \leftarrow States(S_i) \cup \{ q'' \}$ 
19      $Customers(q'') \leftarrow Customers(q') \cup \{ C_i \}$ 
20     pour chaque état  $q \in SubPred(q')$  faire
21       • Suppression de la transition  $q \xrightarrow{e_i} q' : \mathcal{T} \leftarrow \mathcal{T} \setminus \{ q \xrightarrow{e_i} q' \}$ 
22       • Création d'une nouvelle transition  $q \xrightarrow{e_i} q'' : \mathcal{T} \leftarrow \mathcal{T} \cup \{ q \xrightarrow{e_i} q'' \}$ 
23     • Copier les transitions sortantes de l'état  $q'$  vers l'état  $q''$ 
24  $SubPred \leftarrow \emptyset$ 

```

déjà apparu (ligne 1). Si ce n'est pas le cas, il est alors (1) ajouté à \mathcal{C} , l'ensemble des clients observés (ligne 2), (2) associé à l'ensemble des clients de l'état initial (lignes 3 et 4). Ensuite, pour chaque état q associé au client C_i , on vérifie s'il existe une transition de q vers un état q' labellisée par e_i . Si une telle transition existe (lignes 10 et 11), le sous-ensemble $SubPred(q')$ des prédécesseurs de q' est augmenté de l'état q et l'ensemble $SubPred$ est logiquement augmenté de l'état q' ¹. Dans le cas contraire, cela signifie qu'il faudra créer un nouvel état lorsque l'on aura étudié tous les états associés au client C_i . En attendant, l'état puits, q_∞ , permet de se conserver cette information. Ainsi, $SubPred(q_\infty)$ est augmenté de l'état q et l'ensemble $SubPred$ est augmenté de l'état q_∞ (lignes 7 et 8). La dernière étape de l'algorithme (lignes 12 à 23) consiste à vérifier s'il est nécessaire de créer des nouveaux états. Pour cela, chaque état q' appartenant à $SubPred$ est considéré. Si q' est différent de l'état puits et que l'ensemble de ses prédécesseurs est identique à $SubPred(q')$, alors il n'est pas nécessaire de créer un nouvel état car la simple mise à jour des ensembles $States(C_i)$ et $Customers(q')$ (lignes 14 et 15) est suffisante et ne viole pas les contraintes structurelles de l'automate énoncées dans la section 2.2. Sinon, un nouvel état q'' doit être créé (ligne 17). Cette création implique (1) la mise à jour des ensembles $States(C_i)$ et $Customers(q'')$ (lignes 18 et 19), (2), pour chaque état q de $SubPred(q')$, la suppression (si elle existe) de la transition de q vers q' et la création d'une transition de q vers q'' labellisée par e_i (lignes 20 à 22) et (3) de copier toutes les transitions sortant de q' vers q'' (ligne 23). Enfin, l'ensemble $SubPred$ est vidé (ligne 24).

Pour illustrer cet algorithme, nous le déroulons sur les données restreintes à la dimension *Vecteur de communication* issues du troisième batch du cas d'étude. Les figures 5.2 à 5.7 représentent l'automate obtenu après chaque insertion. L'insertion du premier couple, *i.e.* (2, *Facebook*), entraîne la création de l'état q_1 . La transition de q_0 vers q_1 labellisée par *Facebook* permet bien de signifier que la séquence $\langle\langle Facebook \rangle\rangle$ est reconnue par le client 2. Ensuite, un nouveau couple, *i.e.* (2, *Facebook*), est inséré et entraîne la création de l'état q_2 . Nous remarquons qu'il n'a pas été nécessaire de créer une transition entre q_0 et q_2 car la séquence $\langle\langle Facebook \rangle\rangle$ est déjà indexée dans l'automate. Le couple (2, *mail*) est ensuite inséré (figure 5.4). Dans la mesure où il n'existe aucune transition reliant un état associé au client 2 labellisé par l'item *Mail*, un nouvel état q_3 est créé et trois transitions labellisées par *Mail* reliant les états q_0 , q_1 et q_2 à q_3 sont créées. L'insertion du couple (3, *Facebook*) marque l'apparition d'un nouveau client, *i.e.* le client 3. Cependant, dans la mesure où la séquence $\langle\langle Facebook \rangle\rangle$ est déjà indexée dans l'automate, l'insertion de ce couple a pour seule conséquence la mise à jour des clients reconnus par les états q_0 et q_1 . Le cinquième couple à être inséré est (3, *Tweeter*). Aucun des états associés au client 3 ne possède de transition sortante labellisée par *Tweeter*. L'insertion de ce couple entraîne la création de l'état q_4 et des transitions reliant les états associés au client 3 (*i.e.* q_0 et q_1) vers q_4 . Enfin, le couple (3, *Mail*) est

1. Comme indiqué dans le tableau 5.2, l'ensemble $SubPred$ est l'ensemble des états q tel que $SubPred(q)$ est non vide.

inséré. Les états associés au client 3 sont q_0 , q_1 et q_4 . Pour deux d'entre eux (*i.e.* q_0 et q_1), il existe une transition vers q_3 labellisée par *Mail*. Malgré tout, il n'est pas possible de mettre à jour l'état q_3 via l'ajout du client 3. Cela signifierait que toutes les séquences reconnues par q_3 sont reconnues par les clients 2 et 3. Or, parmi elles, la séquence $\langle (Facebook)(Facebook)(Mail) \rangle$ n'est reconnue que par le client 2. Il est alors nécessaire de créer un nouvel état q_5 et de relier les états q_0 et q_1 à ce nouvel état. La figure 5.7 représente l'automate après l'insertion de cet item. Remarquons que cet automate reconnaît bien toutes les sous-séquences de la base de données et qu'il n'existe pas de transition d'un état q vers un état q' telle que $Customers(q') \not\subseteq Customers(q)$.



FIGURE 5.2 – $Insert(2, Facebook)$

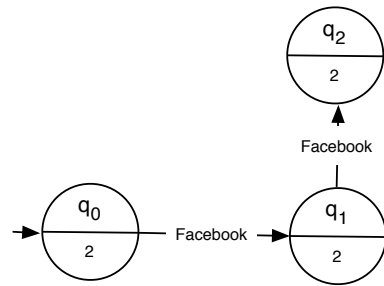


FIGURE 5.3 – $Insert(2, Facebook)$

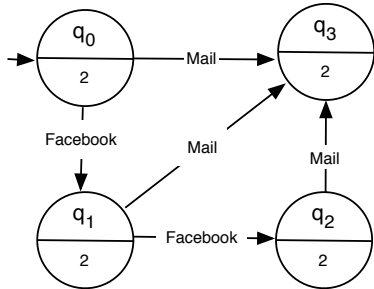


FIGURE 5.4 – $Insert(2, Mail)$

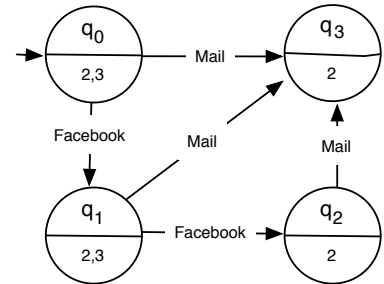


FIGURE 5.5 – $Insert(3, Facebook)$

3.4 Phase 2 : à la recherche des motifs multiniveaux

L'automate généré lors de la première étape ne reconnaît que des séquences de *bas niveau*. Les motifs fréquents reconnus par cet automate n'exploitent donc pas les hiérarchies. En observant la figure 5.1 et en considérant $\sigma = 1$, les motifs séquentiels fréquents reconnus sont $\langle (Mail) \rangle$, $\langle (Facebook) \rangle$ et $\langle (Facebook)(Mail) \rangle$. Pourtant, si l'on observe attentivement *SDB*, nous constatons que la séquence $\langle (Facebook)(Réseaux sociaux)(Mail) \rangle$ est également fréquente et caractérise plus justement la base de données. Nous proposons un parcours en profondeur de l'automate associé à un mécanisme de fusion d'états pour découvrir de telles séquences.

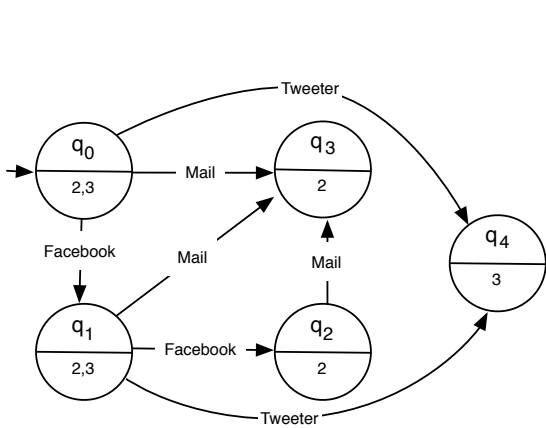


FIGURE 5.6 – $Insert(3, Tweeter)$

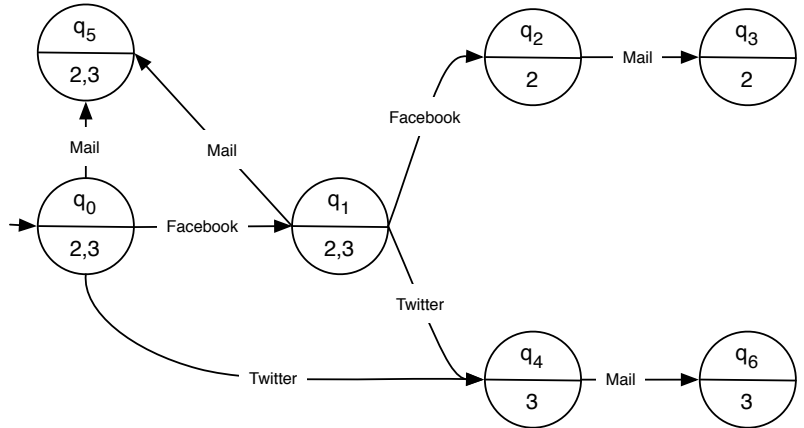


FIGURE 5.7 – $Insert(3, Mail)$

Pour chaque état q de \mathcal{Q} , nous considérons l'ensemble $Succ(q)$ et parcourons l'automate en débutant par l'état initial :

- Les transitions $q \xrightarrow{e_i} q'$ telles que e_i est un item de *bas niveau* et $|Customers(q')|$ (le support des séquences reconnues par q') est supérieur à σ sont conservées. Nous notons $T_{fbn}(q)$ l'ensemble de ces transitions.
- Nous recherchons ensuite le niveau de granularité H_i^j le plus précis (différent des niveaux H_i^{max} et H_i^1) tel qu'il existe au moins un item $e \in Dom(H_i^j)$ où $|\bigcup Customers(q')| \geq \sigma$ avec $q' \in Succ(q, e)$. Nous notons $E_{cand}(q)$ l'ensemble des items qui satisfont la condition précédente. Si $E_{cand}(q)$ est vide, cela signifie qu'il n'existe pas de séquence fréquente préfixée par un élément de $Seq(q)$. Les transitions de $Trans_{out}(q) \setminus T_{fbn}$ sont supprimées de l'automate. Si, à la suite de cette opération, il existe des états qui ne sont plus accessibles, ceux-ci sont supprimés. Dans le cas où E_{cand} n'est pas vide, cela signifie que les séquences $e.e'$ (où $e \in Seq(q)$, $e' \in E_{cand}(q)$ et $.$ est l'opérateur de concaténation de séquences) sont fréquentes. Alors, pour chaque élément de $e' \in E_{cand}(q)$, les états $e'' \in Succ(q, e')$ sont fusionnés. Les transitions de $Trans_{out}(q) \setminus (T_{fbn}(q) \cup (\cup Trans(q, e')))$, avec $e' \in E_{cand}(q)$, sont supprimées. Si, à la suite de cette opération, il existe des états qui ne sont plus accessibles, ceux-ci sont supprimés. Il est important de remarquer que nous recherchons *seulement* le premier niveau de précision où une fusion est possible. En effet, la recherche de motifs séquentiels sur toutes les combinaisons de niveaux possibles est irréalisable en pratique. Cela est dû à un espace de recherche bien plus grand que celui à explorer lors de l'extraction de motifs séquentiels *classiques* (à cause de la combinaison de la multidimensionnalité, des hiérarchies et de la séquentialité).
- Le traitement de l'état q étant terminé, un nouvel état $q' \in Succ(q)$ est choisi et analysé.

A la fin de ce parcours, l'automate obtenu n'indexe que des motifs séquentiels multiniveaux fréquents. Le pseudo-code associé à ce parcours est présenté dans l'algorithme 5.3.

Algorithme 5.3: Traite(q, σ)

```

1 OK ← false ;
2 niveau ← max-1 (max est le nombre de niveaux de précision de la hiérarchie);
3  $T_{cons} \leftarrow \emptyset$  ;
4  $T_{fbn}(q) \leftarrow \{q \xrightarrow{e_i} q'\}$  tq  $e_i$  est un item de bas niveau et  $|Customers(q')| \geq \sigma$  ;
5  $T_{cons} \leftarrow T_{cons} \cup T_{fbn}(q)$  ;
6 tant que niveau > 1 et !OK faire
7    $E_{Cand} \leftarrow \{e | Dom(H^{niveau}) \text{ où } |\bigcup Customers(q')| \geq \sigma \text{ avec } q' \in Succ(q, e)\}$  ;
8   si  $E_{Cand} \neq \emptyset$  alors
9     pour chaque  $e' \in E_{Cand}$  faire
10       FUSION( $Succ(q, e'), e'$ ) ;
11        $T_{cons} \leftarrow T_{cons} \cup Trans(q, e')$  ;
12       OK ← true ;
13   niveau ← niveau - 1 ;
14  $\mathcal{T} \leftarrow \mathcal{T} \setminus T_{cons}$  ;

```

Nous décrivons maintenant le processus de fusion d'états. Nous notons e l'item ayant provoqué la fusion (*i.e.* $e \in E_{cand}(q)$ où q est l'état en cours d'analyse) et E_{fus} l'ensemble des états à fusionner. Cette opération consiste essentiellement à (1) créer un nouvel état, (2) à considérer les transitions entrantes et sortantes des états de E_{fus} et (3) à supprimer les états de E_{fus} qui ne sont pas dans $T_{fbn}(q)$.

- **Création du nouvel état q .** Le nouvel état créé ne possède ni transition entrante ni transition sortante. L'ensemble $Customers(q)$ est défini comme l'union des clients des éléments de E_{fus} ($Customers(q) = \bigcup Customers(q')$ où $q' \in E_{fus}$).
- **Transitions entrantes.** Les transitions entrantes du nouvel état q sont définies de la manière suivante. Une transition $q' \xrightarrow{e} q$ est créée s'il existe une transition $q' \xrightarrow{e_i} q$ où $e_i \subseteq e$. En d'autres termes, l'ensemble des transitions entrantes de q est l'union des transitions entrantes des éléments de E_{fus} étiquetées par e .
- **Transitions sortantes.** L'ensemble des transitions sortantes de q est l'union des transitions sortantes des éléments de E_{fus} . Plus formellement, une transition $q \xrightarrow{e_i} q'$ est créée si il existe une transition $q \xrightarrow{e_i} q'$. A la suite de cette étape, s'il existe des transitions $q \xrightarrow{e_{i_1}} q'_1, \dots, q \xrightarrow{e_{i_k}} q'_k$ telles que $e_{i_1} = \dots = e_{i_k}$, alors les états sont fusionnés. En d'autres termes, s'il

existe des transitions sortantes de q étiquetées par le même item e_i alors les états atteints par ces transitions sont fusionnés entre eux.

- Mis à part les états présents dans T_{fbn} , les états à fusionner sont supprimés (ainsi que les transitions associées).

Le pseudo-code associé à ce mécanisme de fusion est présenté dans l’algorithme 5.4.

Algorithme 5.4: FUSION(Q, e, T_{fbn})

```

1  $Q \leftarrow Q \cup \{q\}$ ;
2  $Customers(q) \leftarrow \bigcup Customers(q')$  où  $q' \in Q$  ;
3  $\mathcal{T} \leftarrow \mathcal{T} \cup \{q' \xrightarrow{e} q\}$  tq il existe  $q' \xrightarrow{e_i} q \in \mathcal{T}$  avec  $e_i \subseteq e$  ;
4 si  $\exists q \xrightarrow{e_{i_1}} q'_1, \dots, q \xrightarrow{e_{i_k}} q'_k$  tq  $e_{i_1} = \dots = e_{i_k}$  alors
5    $\left[ \text{FUSION}(\{q'_1, \dots, q'_k\}, e_{i_1}, T_{fbn}); \right.$ 
6    $\left. \mathcal{T} \leftarrow \mathcal{T} \cup \{q \xrightarrow{e_i} q'\}$  tq il existe  $q' \xrightarrow{e_i} q'' \in \mathcal{T}$  avec  $q' \in Q$ ;
7    $Q_{suppr} \leftarrow Q \setminus T_{fbn}$ ;
8    $\mathcal{T} \leftarrow \mathcal{T} \setminus (Trans_{in}(q') \cup Trans_{out}(q'))$  où  $q' \in Q_{suppr}$ ;
9    $Q \leftarrow Q \setminus Q_{suppr}$ ;

```

Exemple 5.2 Afin d’illustrer le fonctionnement de la méthode d’extraction proposée, nous reprenons l’automate des sous-séquences \mathcal{A} (construit à partir des données du tableau 5.1 restreintes sur la dimension Vecteur de communication) présenté dans la figure 5.1. Nous admettons que le parcours en profondeur est en cours d’analyse de l’état q_0 . L’automate à parcourir est celui généré lors de la première phase et est présenté dans la figure 5.7. Puisque $|Customers(q_1)| = 2$, $|Customers(q_5)| = 2$ et que Mail et Facebook sont des items de bas niveau, nous avons $T_{fbn}(q_0) = \{q_0 \xrightarrow{Facebook} q_1, q_0 \xrightarrow{Mail} q_5\}$. Ensuite, la recherche du niveau le plus précis permettant une fusion aboutit à $E_{Cand} = \{\text{Réseaux sociaux}\}$. Par conséquent, les états q_1 et q_4 doivent être fusionnés. La figure 5.8 illustre l’automate obtenu après le traitement de q_0 .

Nous ne détaillons pas toutes les étapes du parcours de l’automate initial. Malgré tout, la figure 5.9 présente l’automate des séquences fréquentes extraites grâce à notre approche. Comme nous le constatons, parmi ces motifs séquentiels, il en existe un à la fois convergent et divergent : $\langle (Facebook)(Réseaux sociaux)(Mail) \rangle$. Nous rappelons qu’aucune approche existante ne peut extraire de tels motifs.

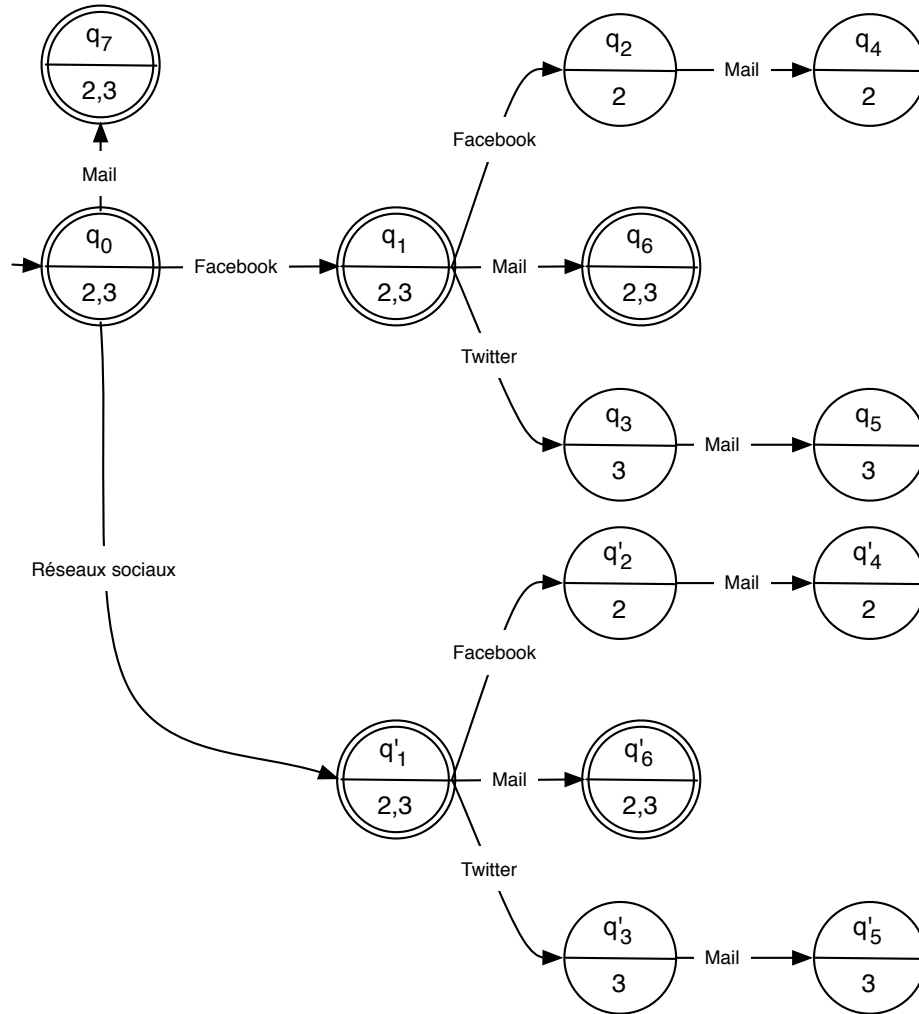


FIGURE 5.8 – Automate des séquences après traitement de q_0

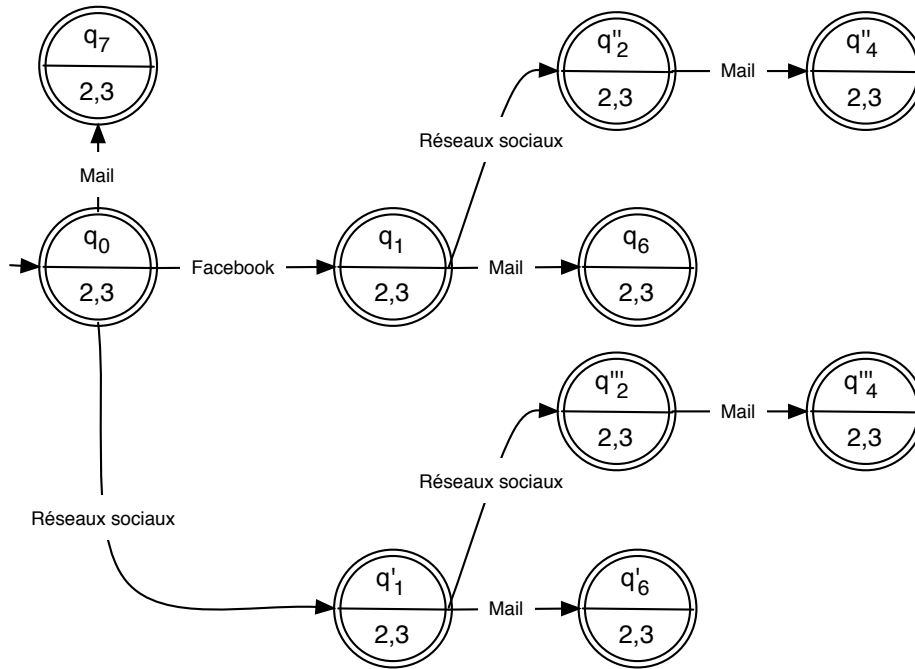


FIGURE 5.9 – Automate des séquences fréquentes de *SDB* restreinte sur la dimension *Vecteur de communication*

3.5 Extension aux données multidimensionnelles

Le début de cette section a été consacré à l'explication de notre approche sur des données décrites sur une seule dimension. Nous discutons maintenant des légères modifications à apporter pour traiter des données multidimensionnelles.

Influence sur la structure de l'automate

L'étape de construction de l'automate des sous-séquences est peu influencée par la gestion de données multidimensionnelles. En effet, il suffit simplement d'étiqueter les transitions par des items multidimensionnels. La méthode de construction ne doit donc pas être modifiée mais l'automate généré contient naturellement plus de transitions et d'états.

Influence sur le processus de fusion d'états

Au cours de cette étape, pour chaque état, l'ensemble de ses successeurs est observé afin de trouver le premier niveau de granularité autorisant une fusion. Lorsque les données sont décrites sur une seule dimension, l'ordre dans lequel ce niveau est recherché est naturellement guidé par la hiérarchie de cette dimension. Dans le cas où les données sont décrites sur plusieurs dimensions, cet ordre est plus délicat à établir car il impacte directement sur les motifs extraits. Par exemple, pour un état q donné, l'ensemble de ses successeurs peut autoriser à la fois une fusion sur le niveau

(*Ville, Catégorie*) et sur le niveau (*Pays, Type*). Face à un espace de recherche immense, il n'est pas concevable de considérer toutes les combinaisons et il est donc nécessaire de déterminer quelle combinaison de niveaux est à considérer en premier. L'ensemble des combinaisons des niveaux de précisions des différentes hiérarchies peut être représenté par un treillis appelé *treillis des cuboïdes*. Le treillis des cuboïdes associé aux hiérarchies H_{Lieu} et $H_{Produit}$ est présenté dans la figure 5.10. Définir l'ordre des niveaux de précision équivaut donc à fixer une stratégie pour parcourir ce treillis. Pour ce faire, nous établissons un ordre $<_D$ sur les dimensions $D_1 <_D D_2 <_D \dots <_D D_N$ signifiant : il est préférable de généraliser d'abord sur D_1 puis sur $D_2 \dots$. Cet ordre $<_D$ est défini par l'utilisateur en fonction de ses besoins d'analyse. Par exemple, si l'on considère $Lieu <_D Vecteur$, l'ordre des cuboïdes considérés sera (*Pays, Vecteur*), (*Ville, CatVecteur*), (*Continent, Vecteur*), (*Pays, CatVecteur*), (*Ville, ALLVecteur*), \dots

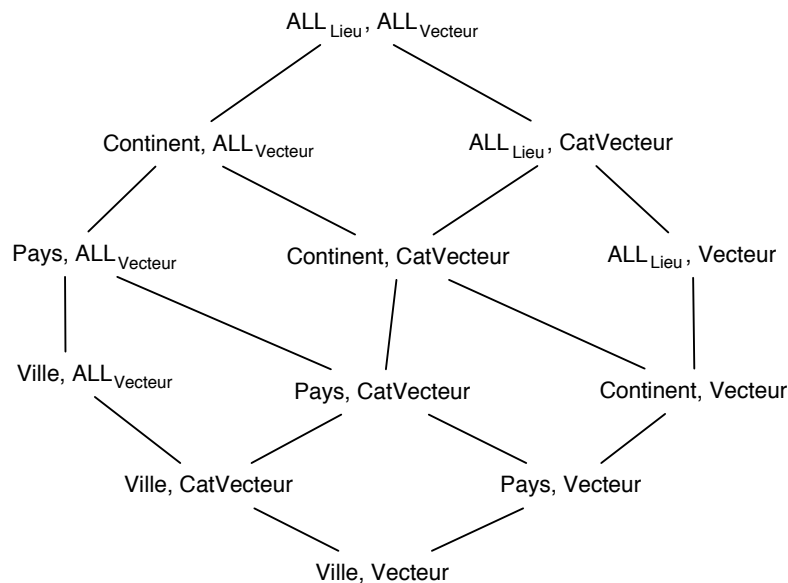


FIGURE 5.10 – Le treillis des cuboïdes associé aux hiérarchies H_{Lieu} et $H_{Vecteur}$

3.6 Expérimentations

Nous présentons ici diverses expérimentations menées sur des données synthétiques et réelles pour valider l'approche *SALINES* proposée dans cette section. Pour cela, nous nous intéressons (1) au nombre de motifs multiniveaux extraits par rapports aux motifs séquentiels fréquents de bas niveaux, (2) au temps d'extraction des motifs et (3) à la mémoire vive consommée. Ces expérimentations ont été menées sur un EeePC 1000 à 1,6Ghz et 1Go de RAM. Les algorithmes proposés ont été implémentés en Java 1.6.

Résultats sur des jeux de données synthétiques

Les données multidimensionnelles ont été générées grâce à un générateur de données aléatoires (suivant une distribution aléatoire uniforme) utilisé pour l'évaluation de méthodes de construction de cubes de données. La convention pour nommer les jeux de données est la suivante : L10kC1kD2Lv2 signifie que la taille de S_{SDB} est 10k, que le nombre de clients est 1k et que les données sont décrites sur 2 dimensions observables sur 2 niveaux de granularité (sans le niveau *ALL*). Nous évaluons l'impact de deux paramètres critiques : le nombre de dimensions et la profondeur de ces dimensions.

Influence du nombre de dimensions L'impact du nombre de dimensions sur les performances globales de l'approche (*i.e.* nombre de motifs extraits, temps d'extraction et consommation mémoire) est maintenant évalué selon différentes valeurs de support minimum. La figure 5.11 présente les résultats obtenus. Une première observation concernant ces résultats est que, bien entendu, l'augmentation du support minimum implique moins de motifs découverts. En outre, il est intéressant de remarquer qu'à partir d'un certain seuil de support minimum ($\sigma = 0,07$), plus aucun motif de bas niveau n'est extrait. Ainsi, passé ce seuil, seuls des motifs généraux sont extraits fournissant ainsi un aperçu représentatif du jeu de donnée. Notons que ces deux remarques sont valables pour toutes les expérimentations présentées ici.

La figure 5.11(b) (resp. figure 5.11(c)) présente le temps d'exécution (resp. la consommation mémoire) sur des jeux de données comportant 5,10 ou 15 dimensions en fonction de différentes valeurs de support minimum. Les caractéristiques du jeu de données sont L20kC100Lv3. Concernant le temps d'extraction, nous observons que celui-ci converge autour de 10 secondes quand le support est supérieur à 0.05. De plus, il est à noter que moins une donnée contient de dimensions, plus le temps d'extraction est élevé. Concernant la consommation mémoire, celle-ci reste bornée autour de 100Mo.

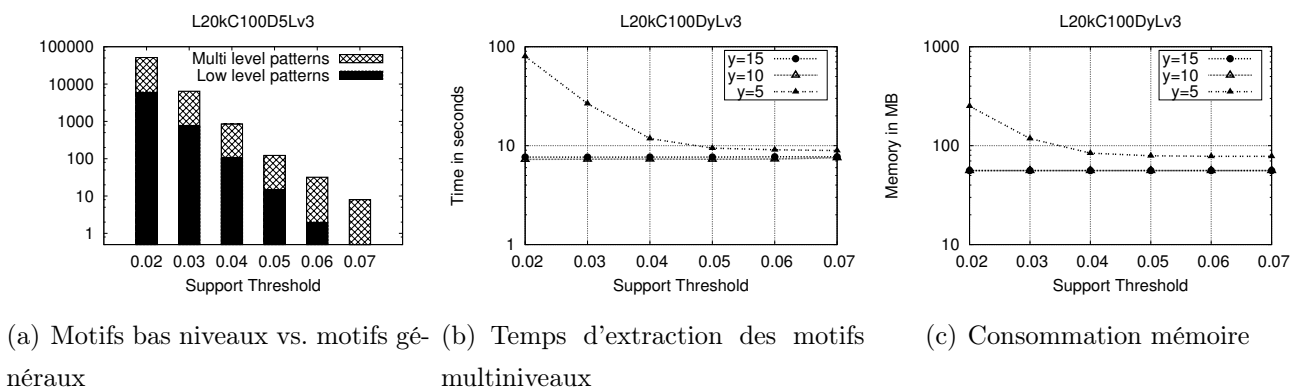


FIGURE 5.11 – Influence du nombre de dimensions (L20kC100Lv3)

Influence de la hauteur des hiérarchies L'impact du nombre de la hauteur des hiérarchies sur les performances globales de l'approche (*i.e.* nombre de motifs extraits, temps d'extraction et consommation mémoire) est maintenant évalué selon différentes valeurs de support minimum. La figure 5.11 présente les résultats obtenus. La figure 5.12(b) (resp. figure 5.12(c)) présente le temps d'exécution (resp. la consommation mémoire) sur des jeux de données dont les dimensions sont décrites sur 3,4 ou 5 niveaux en fonction de différentes valeurs de support minimum. Les caractéristiques du jeu de données sont L20kC100D5. Sur la figure 5.12(b), nous observons que le temps d'extraction est borné autour de 10 secondes quand le support minimum dépasse 0,05. De plus, moins les hiérarchies sont profondes, moins l'extraction est rapide. Ceci s'explique par le fait que un plus grand nombre de fusions d'états est réalisé. Enfin, la figure 5.12(c) permet de conclure que la consommation mémoire est bornée autour de 100 Mo.

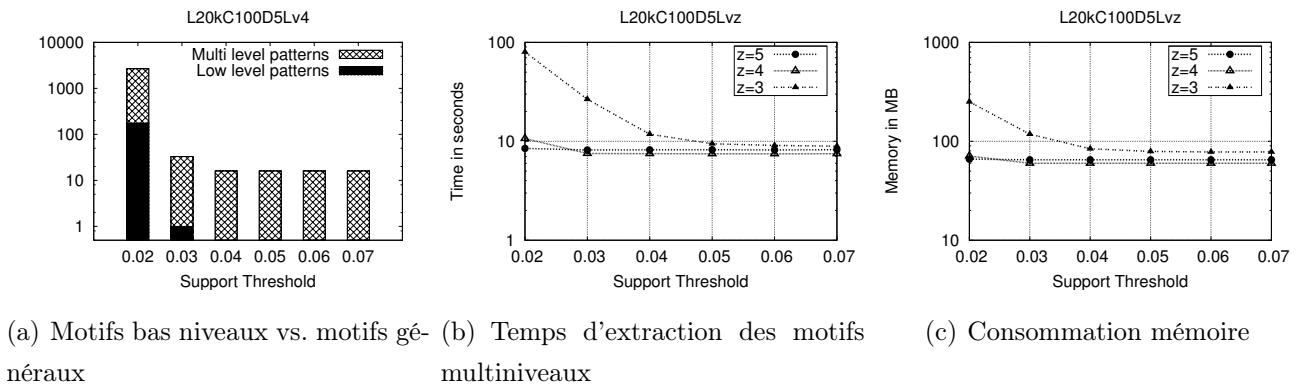


FIGURE 5.12 – Influence de la hauteur des hiérarchies (L20kC100D5)

Ces résultats nous autorisent à dresser quelques conclusions sur la qualité de l'approche *SALINES*. Premièrement, elle permet l'extraction de motifs généralisés quand le support minimum est élevé et fournit donc une vision plus représentative du jeu de données fouillé. Ensuite, ces expérimentations permettent de conclure quant au passage à l'échelle de *SALINES*.

3.7 Résultats sur un jeu de données réelles

Le jeu de données utilisé dans cette série d'expérimentations provient de capteurs implantés sur quatre pompes hydrauliques d'une centrale nucléaire. Chaque item est décrit sur 10 dimensions et représente, pour une pompe donnée, les valeurs émises par les 10 capteurs associés à cette pompe à un instant t . Pour chacune de ces dimensions, nous avons arbitrairement construit sa hiérarchie de hauteur 3 (nous ne comptabilisons pas le niveau *ALL*). Nous considérons chaque pompe comme

un client et avons expérimenté notre approche selon les 3 valeurs de support minimum possibles : 0,5, 0,75 et 1².

La figure 5.13 présente les résultats obtenus. Si l'on considère le nombre de motifs généraux obtenus (*c.f.* figure 5.13(a)), il est possible de conclure de l'intérêt de l'approche *SALINES*. En effet, alors qu'aucun motif de bas niveau ne peut être extrait, notre approche permet de découvrir des séquences généralisées. Concernant le temps d'exécution (*c.f.* figure 5.13(b)), il est notable qu'il demeure stable aux alentours de 200 secondes. Similairement, la consommation mémoire (*c.f.* figure 5.13(c)) est également stable (autour de 600 Mo).

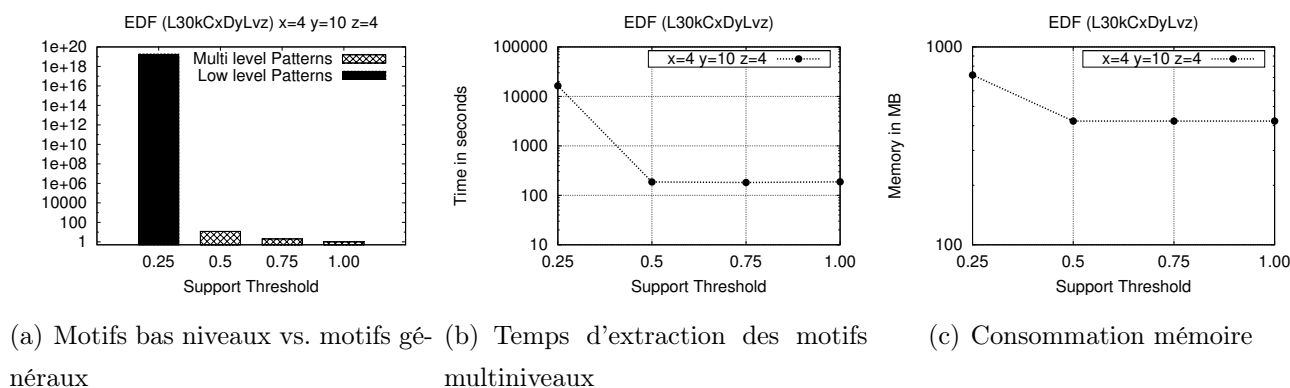


FIGURE 5.13 – Expérimentations sur données réelles

3.8 Premières conclusions

Dans cette section, nous apportons une solution originale basée sur un automate afin d'extraire une nouvelle catégorie de motifs séquentiels multidimensionnels : les motifs séquentiels en "montagne russe". Dans un premier temps, l'automate des sous-séquences de bas niveaux est construit puis parcouru afin d'extraire cette nouvelle catégorie de motifs. Au cours de ce parcours, des fusions d'états sont réalisés pour faire émerger des items fréquents généraux. Les premières expérimentations menées confirment que notre approche est applicable sur des bases de données volumineuses.

Malgré ces bons résultats, l'application immédiate de l'approche *SALINES* pour résumer un flot n'est pas réaliste. Deux arguments étayent cette affirmation. Le processus proposé dans *SALINES* impose de construire l'automate des sous-séquences dans un premier temps. Extraire de telles séquences fréquentes sur l'intégralité du flot est donc impossible à cause de sa taille non bornée. A l'inverse, l'extraction de ces séquences sur des batchs pourrait être envisagée. Cependant, cela se traduirait par l'obtention d'une suite en constante expansion d'ensembles de séquences fré-

2. $\sigma = 0,25$ est inutile car cela implique que toutes les sous-séquences de la base sont fréquentes

quentes et violerait la contrainte de compacité d'un résumé de flot. En outre, bien que satisfaisant dans un contexte statique, le temps d'extraction de telles séquences par *SALINES* risquerait de bloquer le flot de données.

4 ALIZES : des motifs séquentiels pour résumer un flot de données

Dans cette section, nous proposons *ALIZES*, une approche qui s'inspire des mécanismes précédemment décrits pour construire un résumé de flot de données à partir de séquences multidimensionnelles et multiniveaux. Dans un premier temps, nous décrivons son fonctionnement général et poursuivons cette section en détaillant chacune des étapes. Similairement à la description de *SALINES*, nous choisissons de présenter *ALIZES* dans un contexte monodimensionnel et détaillons ensuite les modifications mineures à apporter à l'approche pour autoriser la gestion de données multidimensionnelles. Ensuite, nous montrons comment cette approche peut être parallélisée pour optimiser le temps de traitement. Enfin, quelques résultats expérimentaux obtenus sur des jeux de données réelles et synthétiques sont présentés et permettent de dresser quelques premières conclusions.

4.1 Aperçu général de l'approche

Le processus global de l'approche proposée est illustré sur la figure 5.14 et peut être résumé ainsi. D'abord, l'algorithme *SPAMS* [VSMP09] pour générer un automate indexant les séquences fréquentes d'items de bas niveau pour chaque batch (étape 1). Chacun de ces automates est ensuite simplifié afin de n'indexer que les séquences fermées d'items (étape 2). Ces automates sont ensuite insérés dans une *titled time window* (étape 5). Lorsque la première fenêtre est pleine, les automates la composant sont agrégés (étape 3). Cette agrégation est réalisée en deux temps. D'abord, l'union de ces automates est effectuée. Cet automate est ensuite parcouru en profondeur pour rechercher si des fusions d'états sont possibles. Intuitivement, la fusion permet de faire émerger des séquences multiniveaux qui représentent la plupart des automates fusionnés. Par exemple, considérons une *titled time window* dont la première fenêtre est de taille 2, la hiérarchie associée à la dimension *Vecteur de communication* et supposons que *SMS* a été fréquent dans un batch de la première fenêtre et que *MMS* l'a été dans l'autre batch. Dès lors, *Mobile* a été fréquent pendant l'intervalle de temps représenté par l'union des deux batchs. Ainsi, les états associés à *SMS* et *MMS* seront fusionnées pour produire un nouvel état indexant l'item généralisé *Mobiles*. Lorsque toutes les fusions possibles ont été réalisées, l'automate est alors inséré dans la fenêtre suivante (étape 4). Si

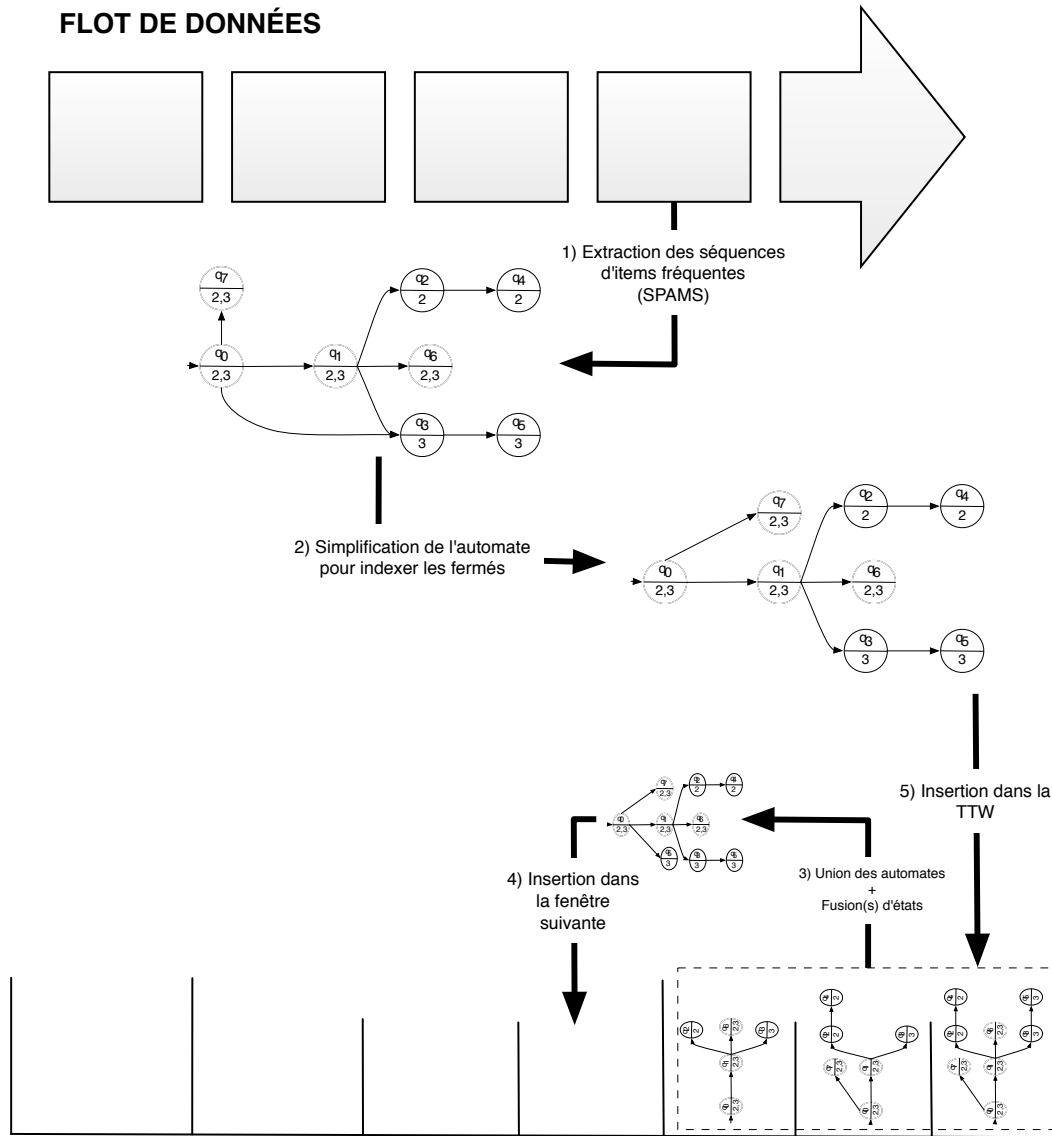


FIGURE 5.14 – Aperçu général de l'approche ALIZES

celle-ci est également pleine, les étapes d'union des automates, de recherche des états à fusionner et du déplacement de l'automate résultat sont répétées jusqu'à l'insertion d'un automate fusionné dans une fenêtre non pleine.

4.2 Extraction des fréquents et obtention des fermés

Soit $F = B_0, B_1, \dots$ une séquence potentiellement infinie de batches. La première étape consiste à extraire les séquences fréquentes d'items de bas niveau. Pour cela, nous utilisons l'algorithme *SPAMS* [VSMP09] pour générer un automate des séquences fréquentes d'items. Nous notons σ ($0 \leq \sigma \leq 1$) le support minimum spécifié par l'utilisateur. La structure et les caractéristiques de

l'automate généré sont développées dans la section précédente. Nous ne les détaillons donc pas ici. Nous notons \mathcal{A}_i l'automate des séquences fréquentes d'items associé au batch B_i .

L'insertion de ces automates dans la *tilted time window* $T = \langle W_1, \dots, W_k \rangle$ implique de les agréger lorsqu'une fenêtre de T est pleine. Ce processus d'agrégation d'automates se décompose en deux étapes : d'abord effectuer l'union de ces automates puis parcourir ce nouvel automate pour le résumer. Bien que nous détaillons cette étape dans la section suivante, il est essentiel de remarquer la chose suivante. L'algorithme classique pour réaliser l'union d'automates déterministes peut rendre l'automate produit non déterministe. Bien qu'une méthode pour rendre déterministe un automate existe, sa complexité en temps est exponentielle sur le nombre d'états et il n'est donc pas envisageable de l'appliquer sur un flot. Il est alors nécessaire de modifier en amont les automates qui seront fusionnés et proposons pour cela de ne considérer que les séquences fermées.

Introduits dans [PBTL99], les motifs fermés sont des motifs tels qu'il n'existe pas de *super motifs* de même support. Les motifs (séquentiels) fermés permettent alors de représenter les connaissances extraites de manière compacte et sans perte d'information. Etudions maintenant leur intérêt dans notre cadre applicatif. Intuitivement, l'obtention des séquences d'items fréquentes fermées à partir d'un automate \mathcal{A}_i est réalisable à faible coût. En effet, considérons q un état quelconque³ d'un automate \mathcal{A}_i . Par définition, les séquences reconnues par cet état (*i.e.* $Seq(q)$) ont le même support et il existe une relation d'inclusion entre ces séquences. Dès lors, parmi les séquences de $Seq(q)$, celle dont la taille est la plus grande est une séquence fermée représentant, sans perte d'information, les autres séquences de $Seq(q)$. D'un point de vue algorithmique, cela équivaut à supprimer toutes les transitions entre deux états q_1 et q_2 telles qu'il existe un chemin de longueur supérieure à 1 entre ces deux états (*i.e.* en d'autres termes, cela équivaut à supprimer toute relation de transitivité entre les états de l'automate). Enfin, pour que l'automate modifié n'indexe que les séquences fermées, il suffit de ne considérer comme finaux que les états q tels que le support associé à cet état soit différent (*i.e.* supérieur) à tous les supports des états accessibles via une transition sortante de q .

Comme nous le prouvons ci-dessous, la topologie d'un automate à l'issue de cette étape est une arborescence. Ce point est particulièrement important car il permet de proposer des solutions efficaces quant à l'agrégation des automates lors du passage à une granularité de temps supérieure dans la *tilted time window*.

Exemple 5.3 Cette méthodologie d'obtention des fermés est illustrée à partir du premier batch, restreint à la dimension Vecteur de communication. Le support minimum, σ , est fixé à $\sigma = \frac{1}{3}$. La

3. Nous rappelons que tous les états de \mathcal{A}_i sont des états finaux car indexant des séquences fréquentes.

figure 5.15 présente l'automate généré par SPAMS. Il existe deux chemins différents pour atteindre l'état q_2 . Le premier, $q_0 \xrightarrow{\text{Facebook}} q_2$ indexe la séquence $\langle (\text{Facebook}) \rangle$ et est de taille 1. Le second, $q_0 \xrightarrow{\text{Twitter}} q_1 \xrightarrow{\text{Facebook}} q_2$ indexe la séquence $\langle (\text{Twitter})(\text{Facebook}) \rangle$ et est de taille 2. Ainsi, les transitions du premier chemin sont supprimées. De plus, tous les états sont finaux dans l'automate initial. Pourtant, les ensembles de clients associés à q_1 et q_2 sont identiques. Dans la mesure où q_1 est un prédécesseur de q_2 , l'état q_1 n'est plus considéré comme final. La figure 5.16 représente l'automate considéré dans cet exemple après son traitement.

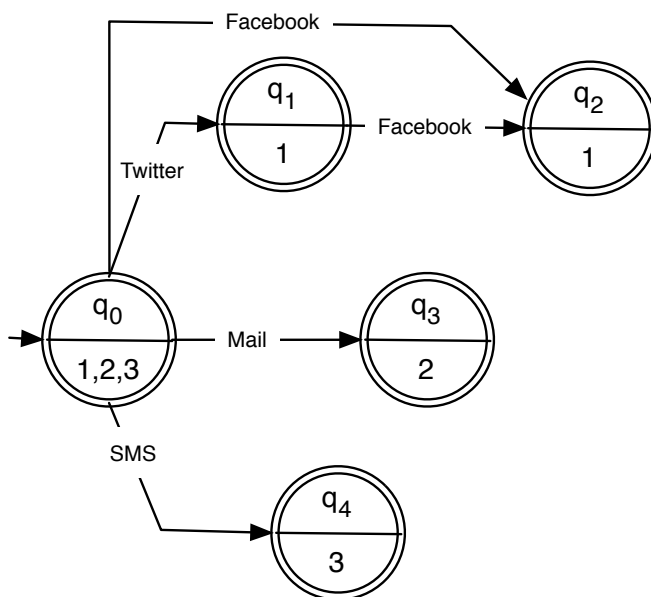


FIGURE 5.15 – L'automate des séquences d'items fréquents du batch 1 réduit sur la dimension *Vecteur de communication* ($\sigma = \frac{1}{3}$)

Nous prouvons maintenant que cette méthodologie transforme un automate \mathcal{A} dont la topologie est celle d'un graphe orienté quelconque en un automate $\tilde{\mathcal{A}}$ dont la topologie est une arborescence.

Propriété 5.1 *L'automate $\tilde{\mathcal{A}}_i$ obtenu à l'issue des transformation décrites ci-dessus a une topologie d'arborescence.*

Preuve Pour prouver qu'un graphe est une arborescence, il suffit de prouver que (1) il existe une racine unique et (2) que quelque soit un nœud (hormis la racine), il possède un et un seul père. L'existence d'une racine unique est triviale et correspond à l'état initial q_0 . Nous nous attachons maintenant à prouver que chaque nœud possède un seul père et raisonnons par l'absurde.

Supposons qu'il existe au moins un nœud possédant au moins deux pères. Pour alléger cette preuve, nous supposons, qu'il existe un seul nœud, noté n , possédant deux pères, notés n_1 et n_2 . Deux cas doivent être étudiés :

1. $Customers(n_1) \cap Customers(n_2) = \emptyset$

Par définition de l'automate manipulé, nous savons que :

$$Customers(n) \subseteq Customers(n_1)$$

$$\text{et } Customers(n) \subseteq Customers(n_2)$$

Par conséquent, $Customers(n) \subseteq Customers(n_1) \cap Customers(n_2)$ ce qui est impossible par hypothèse.

2. $Customers(n_1) \cap Customers(n_2) \neq \emptyset$

Nous notons s_1 la séquence reconnue par l'état n en passant par l'état n_1 et s_2 la séquence reconnue par l'état n en passant par l'état n_2 . Les séquences s_1 et s_2 sont des séquences fermées comme cela est démontré dans la propriété précédente. Pourtant, dans la mesure où s_1 et s_2 sont supportées par les mêmes clients (*i.e.* $Customers(n)$), il existe au moins une séquence s supportée par les mêmes clients et plus longue que s_1 et s_2 . Ceci est impossible puisque nous avons prouvé que l'automate indexe les motifs fermés.

□

A l'issue de ces étapes (*i.e.* extraction des fréquents et obtention des fermés), nous obtenons donc des automates à la topologie d'arborescence qui seront insérés dans la *titled time window* $T = \langle W_1, \dots, W_k \rangle$.

4.3 Union d'automates et réduction

Comme nous l'avons déjà précisé, l'utilisation d'une *titled time window* implique un processus d'agrégation des éléments stockés dans une fenêtre. Dans *ALIZES*, l'originalité provient du fait que ce ne sont pas des valeurs numériques qu'il faut agréger mais des automates. Nous réalisons cette agrégation en deux étapes. dans un premier temps, une union d'automates est réalisée. Ensuite, cet automate est parcouru dans l'objectif de réaliser des fusions d'états permettant l'indexation de séquences multiniveaux. Nous détaillons maintenant chacune de ces étapes.

Union de deux automates à la topologie d'arborescence

Avant de décrire précisément le processus d'union d'arborescences, nous introduisons quelques notations. Nous assimilons volontairement un automate à un graphe orienté valué dont les nœuds

sont les états de l'automate et les arcs sont les transitions labellisées des items. Malgré tout, certaines notations introduites dans la section 2.2 sont utilisées dans la suite car leur transposition à la théorie des graphes est triviale. Soit $\tilde{\mathcal{A}}$ une arborescence et q un nœud de $\tilde{\mathcal{A}}$. La fonction $Root(\tilde{\mathcal{A}})$ désigne la racine de $\tilde{\mathcal{A}}$. La fonction $subTree(q)$ retourne la sous-arborescence de $\tilde{\mathcal{A}}$ dont la racine est q . Nous introduisons également quelques opérations de manipulation de liste. Soient $L = (val_0, \dots, val_m)$ et $L' = (val'_0, \dots, val'_k)$ deux listes d'éléments quelconques. La fonction $Size(L)$ retourne le nombre d'éléments d'une liste (e.g. $Size(L) = m + 1$ et l'opérateur "." désigne l'opération de concaténation de deux listes (i.e. $L.L' = (val_0, \dots, val_m, val'_0, \dots, val'_k)$). Par exemple, $(1, 2).(3, 4) = (1, 2, 3, 4)$. La notation $L[i]$ (avec $0 \leq i < Size(L)$) désigne le $i^{ième}$ élément de L .

Concrètement, les automates sont fusionnés deux à deux à mesure de leur insertion dans une fenêtre de la *titled time window*. Plus précisément, si un automate est inséré dans une fenêtre vide, aucun traitement n'est appliqué. Quand un second automate est inséré dans la fenêtre en question, les deux automates sont fusionnés (tout en conservant les automates initiaux) pour produire un automate temporaire qui sera fusionné avec le troisième automate et ainsi de suite.

L'arborescence obtenue indexe toutes les séquences reconnues par les deux arborescences considérées en entrée. Nous souhaitons conserver une trace des arborescences où sont apparues les séquences. Pour cette raison, chaque état q abrite une liste ($List(q)$ désigne la liste contenue dans q). Ainsi, $List(q)[i]$ représente l'ensemble des clients qui supportent la séquence indexée par l'état q dans l'intervalle de temps concerné par le $(i+1)^{ième}$ arborescence fusionné.

La méthode de fusion de deux arborescences $\tilde{\mathcal{A}}_1$ et $\tilde{\mathcal{A}}_2$ est présentée dans les algorithmes 5.5 et 5.6 et repose sur un parcours en profondeur simultané des deux arborescences. L'algorithme 5.5 initialise ce parcours en indiquant que le premier appel de la fonction fusion se fera sur les racines des deux arborescences. Notons la présence d'un paramètre supplémentaire, T_Liste , qui indique la taille des listes dans les nœuds de $\tilde{\mathcal{A}}_1$. En effet, de part la méthode utilisée, la taille des listes stockées dans les nœuds $\tilde{\mathcal{A}}_2$ est toujours égale à 1. Inversement, mis à part quand le cas où nous fusionnons les deux premières arborescences d'une fenêtre auparavant vide, l'arborescence $\tilde{\mathcal{A}}_1$ est toujours le résultat de fusions préalables. L'algorithme 5.6 présente la fonction principale dans ce mécanisme de fusion. Soient n_1 un nœud de $\tilde{\mathcal{A}}_1$ et n_2 un nœud de $\tilde{\mathcal{A}}_2$. La première étape consiste à réaliser la concaténation des listes associées à ces deux nœuds. Nous partitionnons ensuite, l'ensemble composé de l'union des successeurs de ces deux nœuds en trois catégories et nous décrivons le traitement appliqué à chacune :

- $NotInA_2$ est l'ensemble des nœuds accessibles par un item e qui ne labellise aucun arc sortant de n_2 . Intuitivement, cet ensemble permet d'indexer les séquences reconnues par $\tilde{\mathcal{A}}_1$ mais pas par $\tilde{\mathcal{A}}_2$. Dès lors, pour un nœud n dans $NotInA_2$, aucune des séquences indexées par un nœud se trouvant dans la sous-arborescence de n n'est indexée par $\tilde{\mathcal{A}}_2$. Cela se traduit par l'insertion de l'élément '*null*' dans toutes les listes stockées dans les nœuds des sous-arborescences des nœuds de $NotInA_2$.
- $NotInA_1$ est l'ensemble des nœuds accessibles par un item e qui ne labellise aucun arc sortant de n_1 . Intuitivement, cet ensemble permet d'indexer les séquences reconnues par $\tilde{\mathcal{A}}_2$ mais pas par $\tilde{\mathcal{A}}_1$. Dès lors, pour un nœud n dans $NotInA_1$, aucune des séquences indexées par un nœud se trouvant dans la sous-arborescence de n n'est indexée par $\tilde{\mathcal{A}}_1$. Puisque l'automate $\tilde{\mathcal{A}}_1$ sert de base à la construction de l'automate fusionné, pour chaque nœud n dans $NotInA_1$ nous recopions ce nœud (et sa transition associée) ainsi que les sous-arborescences de b de l'arborescence $\tilde{\mathcal{A}}_2$ dans $\tilde{\mathcal{A}}_1$. Pour tous les nœuds créés, la liste stockée est le résultat de la concaténation d'une liste vide de taille T_{Liste} et de la liste initiale dans $\tilde{\mathcal{A}}_2$;
- $Common$ représente les nœuds indexant des séquences communes aux deux arborescences. Aucun traitement particulier ne doit être fait sur les nœuds de cet ensemble si ce n'est l'appel récursif de la fonction $Fusion$.

Algorithme 5.5: $Init_Fusion(\tilde{\mathcal{A}}_1, \tilde{\mathcal{A}}_2)$

```

1  $n_1 \leftarrow Root(\tilde{\mathcal{A}}_1)$  ;
2  $n_2 \leftarrow Root(\tilde{\mathcal{A}}_2)$  ;
3  $T\_Liste \leftarrow Size(List(n_1))$  ;
4  $Fusion(n_1, n_2, T\_Liste)$ ;

```

Complexité temporelle Nous supposons que la copie d'une sous-arborescence a un coût unitaire⁴. Le pire des cas correspond à celui où aucune des séquences indexées par les deux arborescences est commune. Dans une telle situation, chaque nœud des deux arborescences est parcouru une seule fois. Si l'on note N_1 (resp. N_2) le nombre de nœuds de $\tilde{\mathcal{A}}_1$ (resp. $\tilde{\mathcal{A}}_2$), alors la complexité temporelle est $O(N_1 + N_2)$.

Propriété 5.2 *La fusion de deux arborescences réalisée grâce aux algorithmes présentés précédemment génère une arborescence.*

4. Pratiquement, elle peut en effet être réalisée grâce à une copie de pointeur.

Algorithme 5.6: Fusion(n_1, n_2, t)

```

1 Common ← {n'_1 | n_1  $\xrightarrow{e}$  n'_1, n_2  $\xrightarrow{e}$  n'_2} ;
2 NotInA_2 ← {n'_1 | si n_1  $\xrightarrow{e}$  n'_1 alors  $\nexists$  n_2  $\xrightarrow{e}$  n'_2} ;
3 NotInA_1 ← {n'_2 | si n_2  $\xrightarrow{e}$  n'_2 alors  $\nexists$  n_1  $\xrightarrow{e}$  n'_1} ;
4 List(n_1) ← List(n_1).List(n_2) ;
5 pour chaque n ∈ NotInA_2 faire
6   pour chaque n' ∈ SubTree(n) faire
7     List(n') ← List(n').('null') ;
8 pour chaque n ∈ NotInA_1 faire
9   Création du nœud n' tel que n_1  $\xrightarrow{e}$  n' ;
10  Copie de SubTree(n) sur n' ;
11  pour chaque n'' ∈ SubTree(n') faire
12    List(n'') ← L.List(n'') tel que Size(L) = t et L[i] = 'null' avec 0 ≤ i < t ;
13 pour chaque n'_1 ∈ Common faire
14   Fusion(n'_1, n'_2, t) tel que n_1  $\xrightarrow{e}$  n'_1, n_2  $\xrightarrow{e}$  n'_2 ;

```

Preuve Cette proposition découle trivialement de l'algorithme 5.6.

□

A la recherche de séquences multiniveaux

La deuxième étape de l'agrégation des automates consiste à parcourir en profondeur l'automate produit par l'union afin de fusionner des états. Avant de décrire ce processus, quelques notations et fonctions sont introduites.

W_i désigne la fenêtre de la *titled time window* dont les automates doivent être fusionnés. Soit \mathcal{L} un ensemble de liste. La fonction $MergeListe(\mathcal{L})$ effectue la fusion des éléments de \mathcal{L} comme suit.

Définition 5.4 (*Fusion de liste*) Soient W_i une fenêtre d'une *titled time window* et \mathcal{E} un ensemble de listes de même taille. La fonction $MergeListe$ est définie ainsi :

$$MergeListe(\mathcal{E}) = List_F \text{ tel que } List_F[i] = \bigcup_{L \in \mathcal{E}} L[i]$$

Exemple 5.4 Soit $\mathcal{E} = \{(\{C_1, C_3\}, \{C_1, C_2\}), (\{C_1, C_2\}, \{C_2\})\}$. Nous avons :

$$MergeListe(\mathcal{E}) = (\{C_1, C_2, C_3\}, \{C_1, C_2\})$$

$Repr(L)$ retourne le nombre d'éléments différents de "null" d'une liste L (stockée dans l'état q). Intuitivement, cette fonction sert à comptabiliser le nombre d'automates initiaux qui contiennent la séquence s (avec s est la séquence reconnue par l'état q).

Le critère conditionnant une fusion d'états est le suivant. Pour chaque état q de \mathcal{Q} , nous considérons l'ensemble $Succ(q)$ et notons s la séquence reconnue par q . Intuitivement, nous recherchons e , l'item le plus spécifique, tel que e tel que s . $\langle e \rangle$ (où l'opérateur "." représente l'opérateur de concaténation de séquences) est une généralisation de séquences indexées dans au moins $(\theta \times 100)\%$ (avec $0 \leq \theta \leq 1$ un paramètre utilisateur) des automates initiaux. Plus formellement, nous recherchons le niveau de granularité H_i^j le plus spécifique (mais différent des niveaux H_i^{max} et H_i^1) tel qu'il existe au moins un item $e \in Dom(H_i^j)$ respectant les conditions suivantes :

- Soit \mathcal{E} un ensemble de listes tel que chaque liste $L \in \mathcal{E}$ est stockée dans un état q' appartenant à $Succ(q, e)$. Nous recherchons donc un item e où :

$$\frac{Repr(MergeList(\mathcal{E}))}{Card(W_i)} \geq \theta$$

- $|Succ(q, e)| > 1$

Si un tel item e existe, les états q' appartenant à $Succ(q, e)$ sont fusionnés. Le traitement de l'état q étant terminé, les éléments contenus dans la liste associée sont agrégés⁵. Ensuite, un nouvel état $q' \in Succ(q)$ est choisi et analysé.

Le mécanisme de fusion des états étant identique à celui décrit dans l'approche SALINES, nous ne le détaillons pas et renvoyons le lecteur à l'algorithme 5.4 décrit dans la section 3.4.

Exemple 5.5 *Nous illustrons les processus de fusion et de compression décrits ci-dessus en considérant les deux premiers batchs du cas d'étude. La figure 5.16 (resp. figure 5.17) présente l'arborescence obtenue après l'application de l'algorithme d'obtention des fermés sur les données du premier (resp. second) batch. Le résultat de la fusion de ces deux arborescences est présenté dans la figure 5.18. En observant cette figure, nous pouvons vérifier que la séquence $\langle (Mail)(Twitter) \rangle$ n'est pas indexée dans le premier automate mais l'est dans le second (et est supportée par le client 4). Le résultat du processus de recherche d'états à fusionner est présenté dans la figure 5.19. Si nous nous attardons sur le traitement de l'état q_0 , on peut remarquer qu'il existe bien un item général, Réseaux sociaux, qui possède une spécialisation présente dans les deux automates initiaux. Cette remarque justifie donc la fusion des q_1 et q_5 .*

5. La moyenne est utilisée.

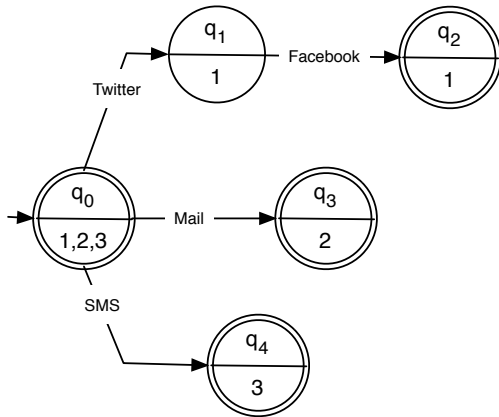


FIGURE 5.16 – Arbre généré à partir des données du premier batch

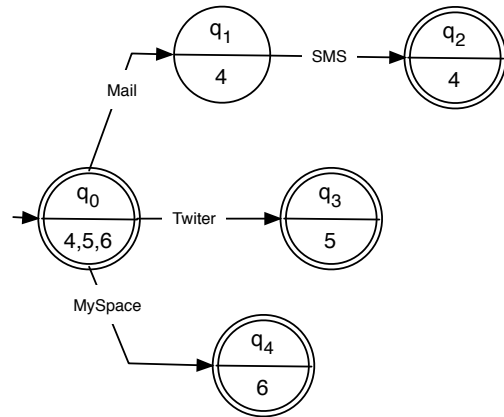


FIGURE 5.17 – Arbre généré à partir des données du second batch

4.4 Extension aux données multidimensionnelles

L'extension de l'approche *ALIZES* au cadre multidimensionnel suit les mêmes règles (*i.e.* labelliser les transitions avec des items multidimensionnels et définir un ordre des cuboïdes) que celles définies pour l'approche *SALINES*. Nous renvoyons le lecteur à la section 3.5 pour obtenir plus de détail concernant cette extension.

4.5 Un processus parallélisable

Jusqu'ici, nous n'avons pas fait de distinction entre les traitements qui doivent nécessairement être réalisés *en ligne* et ceux qui peuvent être réalisés *hors ligne*. Cet aspect est considéré dans cette section. L'extraction des séquences d'items fréquents et l'obtention des motifs fermés doivent, bien entendu, être réalisées *à la volée* (*i.e.* au fur et à mesure que les batchs arrivent). Inversement, l'exécution *en ligne* des autres étapes (fusion des arborescences, recherche des séquences générales et passage à une granularité temporelle supérieure) peut se révéler bloquante (*i.e.* si la durée nécessaire à l'exécution de ces traitements est supérieure à la durée séparant l'arrivée de deux batchs). Nous proposons de considérer ces traitements *hors ligne*. La méthode adoptée est la suivante :

- Un thread principal, P_{main} est exécuté en ligne et gère l'extraction des séquences fréquentes (SPAMS), l'obtention des fermés et l'insertion de cet automate dans la première fenêtre de la *titled time window* ;
- Un thread, P_i , est associé à chaque fenêtre W_i de la *titled time window*. Le thread P_i réalise les fusions d'automate au fur et à mesure que les automates sont insérés dans la fenêtre W_i . Lorsque celle-ci est pleine, P_i exécute la fonction de généralisation et insère l'automate résultant dans la fenêtre suivante (*i.e.* W_{i+1}). Cette insertion réveille alors le thread P_{i+1} .

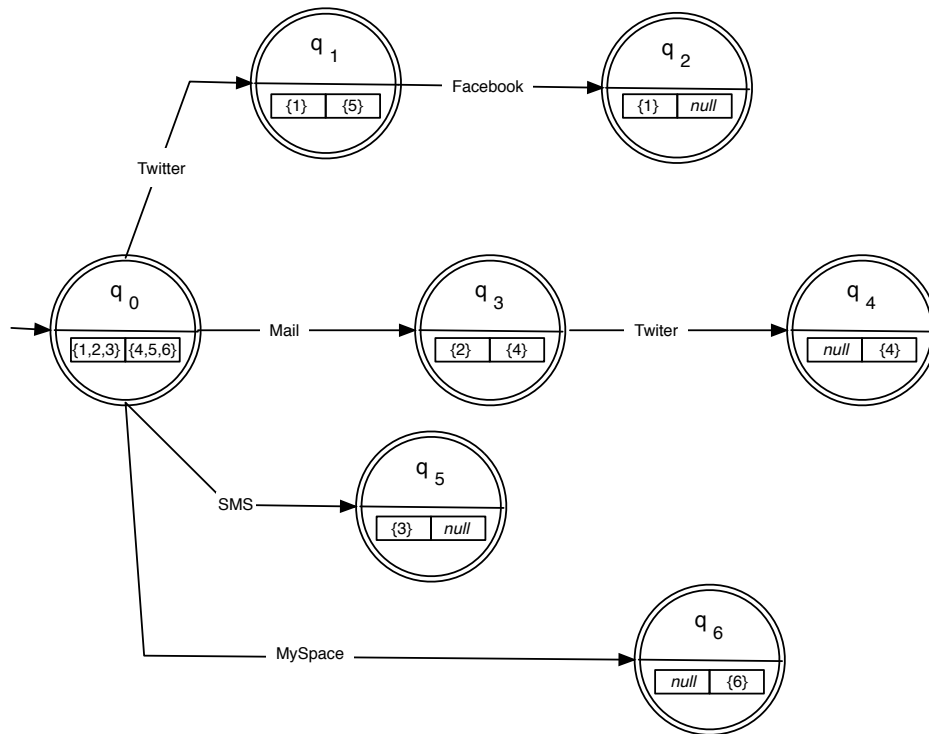


FIGURE 5.18 – Résultat de l'étape de fusion

La distinction entre traitements *en ligne* et *hors ligne* allège la charge du thread principal et permet ainsi de satisfaire le critère de mise à jour rapide du résumé. Néanmoins, le risque d'une telle parallélisation est la non-satisfaction du critère de taille bornée du résumé. En effet, supposons que le thread P_1 n'ait pas le temps de réaliser les traitements qui sont à sa charge car le temps total pour extraire les fréquents et obtenir les fermés nécessaires pour remplir la première fenêtre soit plus rapide que le temps nécessaire à P_1 pour fusionner les automates et exhiber les séquences générales. Deux solutions sont imaginables. La première serait de bloquer le flot pour permettre à P_1 d'achever son traitement. Bien que respectant le critère de taille bornée du résumé, cette solution n'est clairement pas envisageable. La seconde solution, plus réaliste, serait de considérer une liste d'attente d'automates de telle sorte que les automates à fusionner et généraliser s'empileraient sans bloquer le flot. Nous adoptons cette solution et vérifions expérimentalement que ce choix permet tout de même de respecter le critère de taille bornée du résumé.

4.6 Expérimentations

Dans cette section, nous évaluons notre approche sur des jeux de données synthétiques afin de valider son applicabilité dans un contexte de flots de données. A travers ces expérimentations, nous nous sommes donc attachés à montrer que (1) les différentes étapes étaient réalisées efficacement et qu'elles n'étaient pas bloquantes pour le flot et (2) que le critère de compacité était satisfait.

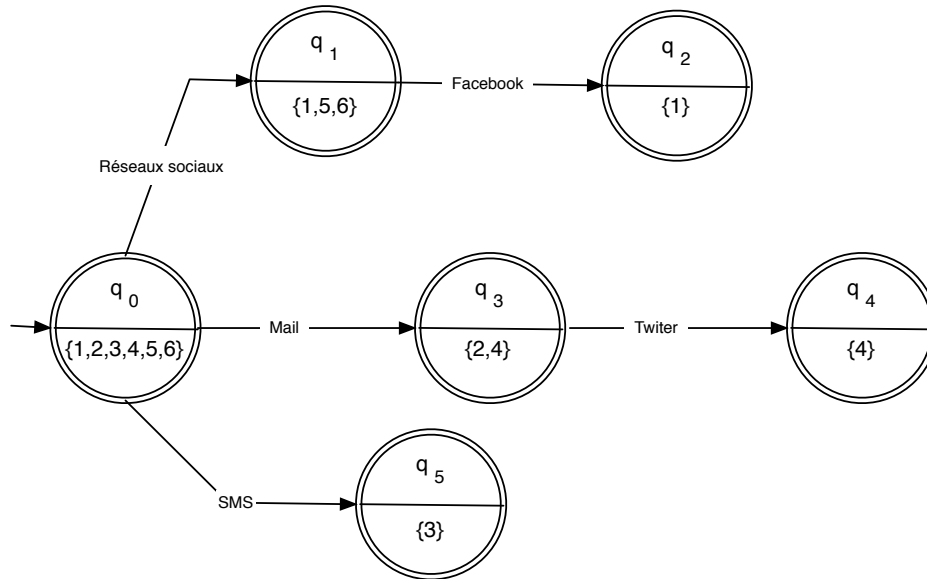


FIGURE 5.19 – Résultat de l'étape de recherche des séquences multiniveaux

Nous proposons alors trois types de résultats. Tout d'abord, nous montrons que la parallélisation de cette approche et l'adoption d'une liste d'attente pour stocker les automates à fusionner et généraliser n'a pas pour conséquence de faire grossir indéfiniment la taille du résumé. Pour affirmer le respect du critère de compacité, nous indiquons l'évolution de la consommation mémoire. Nous montrons ensuite que les étapes réalisées *en ligne* sont réalisées rapidement. Enfin, nous quantifions la compression engendrée par la généralisation d'un automate fusionné lors du passage à une granularité temporelle supérieure en comparant le nombre d'états des automates fusionnés par rapport et le nombre d'états après l'étape de généralisation.

Les jeux de test synthétiques proviennent d'un générateur de batches de données multidimensionnelles aléatoires satisfaisant certaines contraintes (nombre de dimensions, degré des hiérarchies, etc.). La convention pour nommer les jeux de données est la suivante : D4L3C5B100T20 σ 50 θ 75 signifie qu'il y a 4 dimensions avec 3 niveaux de précision chacune, que les arbres des hiérarchies sont de degré 5, que les tests ont été réalisés sur 100 batches de 20K n-uplets et que les paramètres de l'approche valent $\sigma = 50\%$ et $\theta = 75\%$. La *titled time window* utilisée est $T = \langle 4, 4, 4, 4, 2 \rangle$. Ces expérimentations ont été menées sur un EeePC 1000 à 1,6Ghz et 1Go de RAM. Les algorithmes proposés ont été implémentés en Java 1.6.

Le premier critère évalué est la mise à jour rapide des résumés. Pour cela, nous évaluons le temps nécessaire à la réalisation des tâches *en ligne* (*i.e.* construction de l'automate des fréquents et obtention des séquences fermées) par batch. Le jeu de données utilisé est L4C5B100T1 σ 5 θ 75.

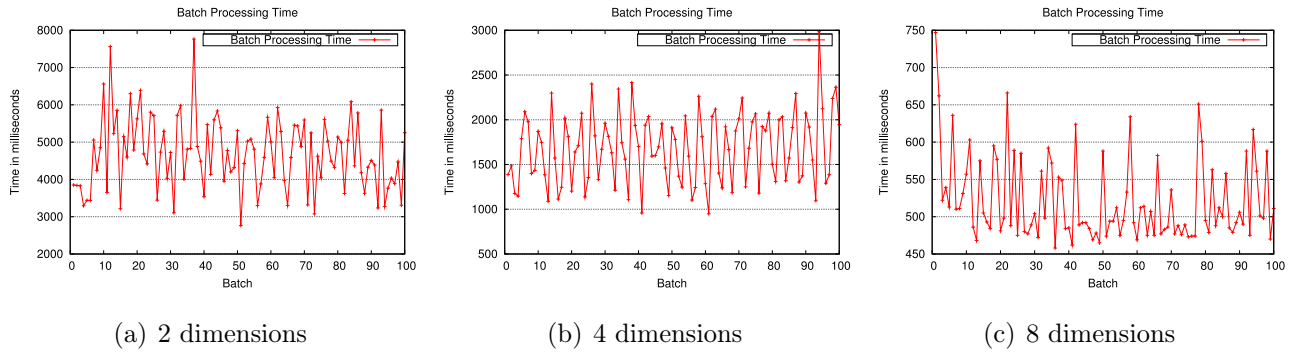


FIGURE 5.20 – Influence du nombre de dimension sur le temps d’extraction et d’obtention des fermés

La figure 5.20 reporte ces temps en fonction de différents nombres de dimensions (2, 4 ou 8). Il est notable que plus il y a de dimensions, moins le temps de traitement est long. Cela s’explique par le caractère aléatoire des données. Nous rappelons que cette étape ne permet que l’extraction de séquences fréquentes de bas niveau. Les données étant générées aléatoirement, le nombre d’items de bas niveau différents est d’autant plus grand qu’il y a de dimensions. Par conséquent, plus il y a de dimensions, moins l’automate généré contient d’état et plus l’étape d’obtention des fermés est rapide.

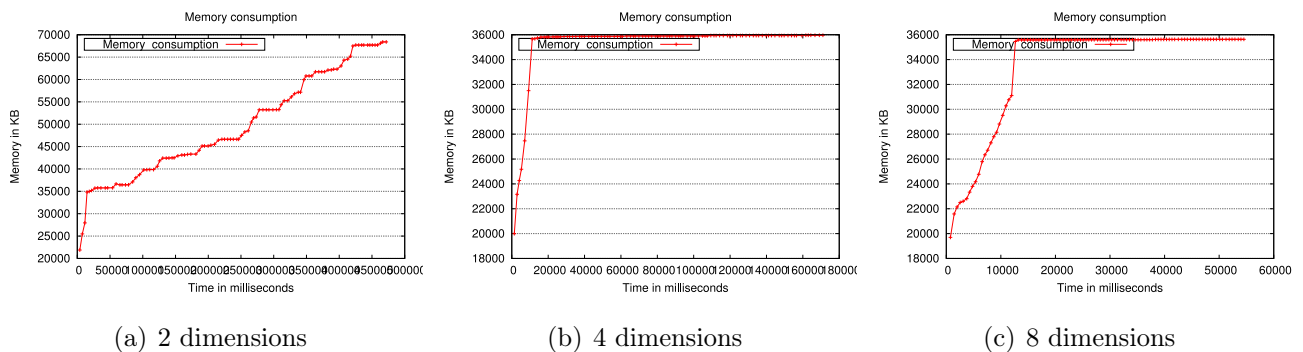


FIGURE 5.21 – Influence du nombre de dimension sur la taille des automates

Le deuxième critère testé est la compacité de la structure proposée. Pour cela, nous évaluons la consommation mémoire lorsque le nombre de dimensions varie (2, 4 ou 8). Le jeu de données utilisé est L4C5B100T1σ5θ75. La figure 5.21 reporte les résultats obtenus. Similairement à la précédente expérimentation, les performances obtenues avec deux dimensions sont moins bonnes que dans les autres cas. Les raisons évoquées précédemment prévalent également ici. Néanmoins, cette expérimentation montre que la taille de la structure est bornée dans tous les cas.

Enfin, nous évaluons la compression réalisée lors de l’étape de généralisation de l’automate issu d’une fusion. Pour cela, nous évaluons la taille des automates avant et après cette étape de

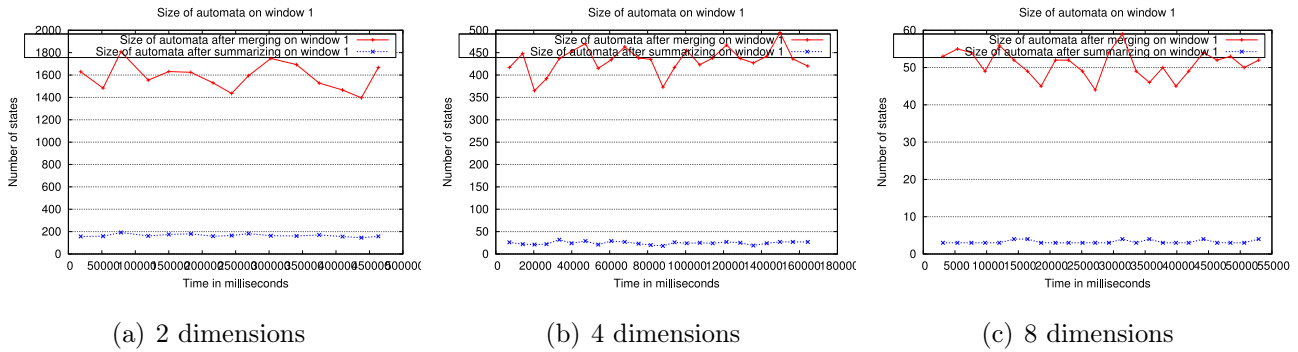


FIGURE 5.22 – Influence du nombre de dimension sur consommation mémoire

généralisation. Notons que dans cette expérimentation, nous considérons seulement la première fenêtre de la *titled time window* (*i.e.* la fusion concerne les automates stockés dans la première fenêtre). Le jeu de données utilisé est L4C5B100T1 σ 5 θ 75. La figure 5.22 reporte les résultats obtenus selon différents nombres de dimensions (2, 4 ou 8). La courbe rouge indique le nombre d'états de l'automate fusionné et la courbe bleue indique le nombre d'états de l'automate après l'étape de généralisation. Tout d'abord, cette expérimentation permet de vérifier que le nombre de motifs fréquents est d'autant plus nombreux que le nombre de dimensions manipulées est faible. Ensuite, cette opération de généralisation permet une compression sémantique efficace de l'automate puisque l'automate obtenu contient environ dix fois moins d'états (et cela indépendamment du nombre de dimensions considérées).

4.7 Conclusions sur ALIZES

Dans cette section, nous avons présenté *ALIZES* une approche de résumé de flots de données grâce à des séquences multidimensionnelles et multiniveaux. Pour cela, nous proposons un processus en plusieurs temps. Tout d'abord, un automate indexant les séquences fréquentes de bas niveau par batch est calculé grâce à l'algorithme *SPAMS*. Cet automate est ensuite simplifié afin de n'indexer que les séquences fermées. Ces automates, dont la topologie est une arborescence, sont insérés dans une *titled time window*. Lorsqu'une fenêtre est pleine, les automates la composant sont agrégés. Cette agrégation est réalisée en deux temps. Nous réalisons dans un premier temps l'union de ces automates. Ensuite, cette automate fusionné est parcouru afin de faire émerger des séquences multiniveaux représentatives. Nous avons montré que ce processus est parallélisable et qu'ainsi, l'approche *ALIZES* peut tout à fait être appliquée sur un flot de données. Les différentes expérimentations menées ont montré que cette approche respecte les critères de compacité et de mise à jour rapide de la structure de résumé. Le respect du critère de représentativité est plus délicat à mesure dans cette approche. D'une certaine manière, le support minimum peut être considéré comme une mesure de représentativité. Néanmoins, si l'on entend par "*mesure de*

représentativité” un score permettant de calculer l’erreur engendrée par le résumé par rapport au flot d’origine, le support minimum n’est pas suffisant. Le calcul d’une telle mesure impliquerait de pouvoir recréer le flot à partir du résumé. Cela est impossible en pratique car il serait extrêmement coûteux de retrouver les occurrences des apparitions des motifs extraits. Cette remarque est d’autant plus vraie dans un contexte de flot. Ainsi, l’utilisation d’une telle technique de résumé doit être réfléchie par l’utilisateur qui doit clairement identifier ses besoins d’analyse et s’assurer de l’adéquation entre ses besoins et la sémantique associée aux motifs fréquents.

Parmi les perspectives associées à ce travail, l’utilisation directe d’un algorithme d’extraction des séquences fermées tel que [CWYL09] permettrait d’éviter la simplification de l’automate généré par *SPAMS*. Nous présentons quelques perspectives conjointes aux deux approches proposées dans ce chapitre dans la section suivante.

5 Discussion

Dans ce chapitre, nous proposons deux approches partageant l’objectif de générer des séquences multidimensionnelles multiniveaux mais appliquées à des contextes différents. *SALINES* permet la découverte de telles séquences au sein de bases de données statiques alors qu’*ALIZES* fait émerger ces séquences multiniveaux pour résumer un flot de données.

Une perspective à court terme et commune aux deux approches serait de les étendre au cadre général des motifs séquentiels (*i.e.* des séquences d’itemsets). Pour *SALINES*, cela impliquerait quelques aménagements de l’algorithme de construction de l’automate des sous-séquences et de la méthode de fusion. Pour *ALIZES*, dans la mesure où l’algorithme *SPAMS* permet l’extraction des motifs séquentiels, seule la méthodologie pour agréger les automates devrait être reconsidérée.

En outre, il pourrait être pertinent de reconsidérer l’utilisation de l’algorithme *SPAMS* dans *ALIZES*. En effet, même si nous avons vu qu’il n’était pas possible d’utiliser *SALINES* comme algorithme d’extraction des séquences fréquentes par batch, il est dommage de ne pas exploiter la multidimensionnalité et les hiérarchies dès la première étape de l’approche *ALIZES*. Les séquences extraites seraient alors plus représentatives du batch et améliorerait la qualité globale du résumé produit.

Chapitre 6

Un modèle conceptuel pour les entrepôts de données contextuels

1 Introduction

Dans les précédents chapitres, nous avons proposé plusieurs stratégies pour exploiter les hiérarchies associées aux dimensions afin de résumer un flot de données. En marge de cette réflexion, nous nous sommes également interrogés sur la pertinence d'utiliser des hiérarchies de mesures dans notre contexte. Dans un environnement statique, l'idée de regrouper les cellules partageant des mesures égales ou similaires pour produire des cubes compacts a été à l'origine des concepts de cubes clos [XSHL06] et de cubes quotients [LPH02]. Dès lors, l'utilisation de mécanismes similaires appliqués à des généralisations de mesures permettrait sans doute de compresser plus efficacement un cube de données.

Cette idée nous a conduits à nous interroger sur la méthode la plus adéquate pour définir des hiérarchies sur des attributs numériques. Une première stratégie consisterait à définir des intervalles de taille de plus en plus grande au fur et à mesure que l'on s'élève dans la hiérarchie. Bien qu'intuitive, cette stratégie nous a paru inadaptée car (1) la définition des intervalles n'est pas triviale et (2) la sémantique associée à un intervalle de valeurs est délicate à exprimer. Une seconde stratégie consisterait alors à définir un ensemble de labels textuels par niveau pour sémantiser la généralisation d'une mesure numérique. Par exemple, les labels associés à la généralisation directe des mesures exprimant la quantité de messages envoyés pourrait être *très peu*, *peu*, *moyen*, *assez élevé*, *très élevé* et les labels associés au niveau : *faible*, *normal* et *élevé*. L'utilisation de telles hiérarchies apparaît judicieuse car elle permettrait d'obtenir des résumés sémantiquement plus riches que dans le cas d'une utilisation de hiérarchies d'intervalles (*i.e.* le nombre de messages envoyés

depuis la Tunisie via les réseaux sociaux fut très élevé entre t_0 et t_1 puis est revenu à la normale).

Nous nous sommes donc intéressés à la définition de ces hiérarchies et plus particulièrement à comment définir correctement les labels associés aux généralisations immédiates d'attributs numériques. En tentant de répondre à des questions telles que “*un nombre de messages envoyés moyen représente-t-il la même quantité pour des adolescents, des adultes ou des personnes âgées ?*”, il est apparu que ces généralisations ne dépendent pas que de la mesure. Par exemple, 10 messages envoyés via facebook par une personne âgée peut être considéré comme un nombre élevé alors qu'il serait moyen (voire faible) pour un adolescent. A notre connaissance, ces hiérarchies, que nous qualifions *contextuelles* n'ont pas été définies malgré qu'elles permettraient la modélisation de nombreuses situations réelles. En effet, dans le domaine du décisionnel, on parle généralement de contexte d'analyse pour désigner le cadre multidimensionnel qui constitue le cadre d'analyse des faits. Autrement dit, par ce terme de contexte, il est sous-entendu que les faits (à travers les valeurs que prennent les mesures) sont fonction des valeurs prises par les dimensions. Or, dans la réalité des données, il s'avère que les mesures elles-mêmes peuvent être hiérarchisées et que les valeurs des attributs caractérisant cette hiérarchisation de mesure peuvent elles-mêmes dépendre d'un certain contexte.

Nous venons de le voir, le scénario considéré dans ce manuscrit peut contenir une telle hiérarchie contextuelle. Malgré tout, nous choisissons, dans ce chapitre, d'illustrer notre proposition avec un exemple plus représentatif de la nécessité de modéliser ce type de hiérarchies : un entrepôt de données médicales. Considérons alors un entrepôt enregistrant les paramètres vitaux (*e.g.*, le pouls, la tension artérielle...) des patients d'un service de réanimation ainsi que les médicaments prescrits à ce patient. Afin de réaliser un suivi efficace des patients, un médecin souhaiterait par exemple connaître ceux qui ont eu une tension artérielle basse au cours de la nuit. Pouvoir formuler ce type de requête suppose l'existence d'une hiérarchie sur la tension artérielle dont le second niveau serait une catégorisation de la tension artérielle (*e.g.* *basse, normale, élevée*). Toutefois, cette catégorisation est délicate car elle dépend à la fois de la tension artérielle mesurée mais aussi de certaines caractéristiques physiologiques (âge du patient, fumeur ou non, ...). Dès lors, une même tension peut être généralisée différemment selon le contexte d'analyse considéré. Par exemple, 13 est une tension *élevée* chez un *nourrisson* alors qu'il s'agit d'une tension *normale* chez un *adulte*. Dans une perspective de détection d'anomalies, la modélisation et l'utilisation de ces hiérarchies contextuelles prend alors tout son sens.

Dans ce chapitre, nous abordons la problématique de la modélisation et de l'implémentation de ces hiérarchies *contextuelles* dans un entrepôt de données en nous posant les quatre questions sui-

vantes : comment modéliser des hiérarchies contextuelles associées à des mesures ou des attributs, (2) comment stocker efficacement cette connaissance, (3) comment l'exploiter afin d'accroître les possibilités d'analyse offertes au décideur et (4) comment la gérer (mise à jour, suppression ou ajout) facilement.

Afin de répondre à ces objectifs, nous proposons une modélisation conceptuelle multidimensionnelle d'un *entrepôt contextuel* (*i.e.* un entrepôt où il existe au moins une hiérarchie contextuelle). Afin de faciliter les interactions entre le concepteur et les futurs utilisateurs de tels entrepôts, nous fournissons également une modélisation graphique associée au modèle conceptuel. Nous détaillons ensuite les méthodes permettant d'exploiter ces hiérarchies (*i.e.* permettre la navigation à travers les opérations de généralisation et spécialisation). Nous nous intéressons ensuite à la mise en œuvre de ces entrepôts et considérons particulièrement comment modéliser la connaissance experte du domaine d'application afin de représenter ces hiérarchies contextuelles. Pour cela, nous faisons le choix d'une base de connaissances externe qui permet de stocker (*i.e.* les connaissances associées à différentes hiérarchies contextuelles sont représentées de la même façon) et efficacement ces connaissances. Enfin, nous présentons une interface web pour interagir facilement avec cette base de connaissance.

La suite du chapitre est organisée de la façon suivante. La section 2 expose un cas d'étude sur des données médicales qui permettra d'illustrer le problème posé et le modèle proposé. Nous discutons dans la section 3 de l'inadéquation des différentes solutions existantes par rapport à la problématique de ce chapitre. La section 4 présente le modèle formel et le modèle graphique que nous proposons pour représenter les entrepôts de données contextuels. Dans la section 5, nous décrivons comment exploiter cette connaissance en nous appuyant sur le cas d'étude précédemment présenté. Au cours de la section 6, nous décrivons comment mettre en œuvre un tel entrepôt. Enfin, nous concluons et indiquons les perspectives à ce travail dans la section 7.

2 Cas d'étude

Pour illustrer la problématique liée à notre approche, nous considérons le cas d'un entrepôt de données enregistrant, pour chaque patient d'un service de réanimation, sa tension, son pouls ainsi que les médicaments prescrits au fil du temps. Nous supposons que ces valeurs, produites par des capteurs, sont correctement intégrées dans l'entrepôt (*e.g.* prise en compte des données manquantes [PSV⁺07]). Chaque patient possède un identifiant unique et est décrit par son nom, son âge, son sexe, la ville où il habite et par un attribut *Fumeur* qui indique si le patient fume. L'âge du patient peut être considéré selon trois niveaux de précision différents. Le premier, *Âge*, in-

dique l'âge du patient. Le second, *SubCatAge*, prend ses valeurs dans l'ensemble {*Nourrisson, Jeune enfant, Enfant, Adolescent, Jeune adulte, Adulte, 3^{ème} âge, 4^{ème} âge*}. Enfin, le niveau le plus général, *CatAge*, prend ses valeurs dans l'ensemble {*Bébé, Jeune, Agé*}.

Afin d'assurer un suivi efficace des patients du service, il est souhaitable de pouvoir formuler des requêtes telles que “*Quels sont les patients dont la tension artérielle a été élevée pendant la nuit ?*” ou “*Quels sont les patients qui se sont vu prescrire une quantité trop importante de médicament X ?*” ou encore “*Quelle est la tension moyenne des enfants ?*”.

Malheureusement, les modèles traditionnels d'entrepôts ne permettent pas de répondre à de telles requêtes pour deux raisons. Premièrement, la notion de tension (resp. posologie) élevée peut être considérée comme une généralisation de la tension mesurée (resp. de la quantité prescrite de médicament). Dans la mesure où les modèles classiques ne permettent pas d'établir une hiérarchie sur les mesures, ces requêtes ne peuvent être formulées. De plus, même si l'on suppose que de telles requêtes sont formulables, la généralisation correcte de valeurs numériques est bien souvent contextuelle.

En effet, dans ce cas d'étude, nous considérons que les notions de *tension élevée*, de *posologie élevée* ou de *patient jeune* sont directement liées à certaines caractéristiques des patients et/ou des médicaments prescrits. Par exemple, un bébé ne doit pas recevoir la même quantité d'un médicament qu'un adulte. Ainsi, une même posologie pourra être considérée comme *faible, normale* ou *élevée* selon l'âge du patient considéré. Une connaissance experte est alors nécessaire pour (1) définir quels sont les attributs qui impactent sur la généralisation d'une mesure et (2) décrire cette généralisation en fonction des valeurs prises par ces attributs.

Dans la suite de ce chapitre, nous nous focalisons sur la catégorisation d'une tension pour illustrer l'approche proposée. Le tableau 6.1 présente quelques exemples de connaissances expertes sur la catégorisation d'une tension en fonction des attributs *SubCatAge* et *Fumeur* d'un patient. Par exemple, une tension à 13 est normale chez un adulte fumeur mais est élevée chez un nourrisson. Cependant, la généralisation d'un âge au niveau *SubCatAge* peut, elle aussi, être contextuelle. Nous supposons dans ce cas d'étude qu'elle dépend à la fois de l'âge mais aussi de la valeur associée à l'attribut *Fumeur*¹. Le tableau 6.2 présente d'ailleurs un extrait des règles permettant une généralisation correcte. Dès lors, généraliser correctement les tensions mesurées implique au préalable d'avoir correctement généralisé l'âge du patient au niveau *SubCatAge*.

1. Typiquement, la tension artérielle *normale* d'un fumeur est plus élevée que celle d'un non-fumeur.

SubCatAge	Fumeur	Tension	CatTension
Nourrisson	Oui ou Non	>12	Élevée
Adulte	Oui	>14	Élevée
3 ^{ème} âge	Oui ou Non	> 16	Élevée
Nourrisson	Oui ou Non	Entre 10 (inclus) et 12 (inclus)	Normale
Adulte	Oui	Entre 12 (inclus) et 14 (inclus)	Normale
...

TABLE 6.1 – Exemple de règles expertes décrivant la catégorie d'une tension (CatTension) en fonction de la tension mesurée, de la classe d'âge d'un patient (SubCatAge) et de l'attribut Fumeur.

Age	Fumeur	SubCatAge
Entre 25 et 70 (inclus)	Non	Adulte
Entre 25 et 65 (inclus)	Oui	Adulte
...

TABLE 6.2 – Exemple de règles expertes décrivant la généralisation au niveau *SubCatAge* en fonction de l'âge du patient et de l'attribut Fumeur.

Dans la prochaine section, nous discutons des raisons qui rendent impossible la prise en compte de ces spécificités par les solutions actuelles. Nous introduisons ensuite un modèle conceptuel d'entrepôt ainsi que son formalisme associé pour pouvoir communiquer efficacement avec les experts du domaine. Enfin, nous proposons des solutions efficaces et génériques pour stocker et exploiter ces hiérarchies contextuelles.

3 Panorama des travaux existants

Au cours de cette section, nous survolons quatre thématiques de recherche connexes à la problématique étudiée dans ce chapitre. Dans un premier temps, un panorama des types de hiérarchies proposées jusqu'ici est dressé. Nous nous intéressons ensuite aux modèles conceptuels d'entrepôts existants dans la littérature. Dans la mesure où nous désirons introduire de la flexibilité dans la conception et l'analyse d'un entrepôt de données, nous nous intéressons aux approches qui partagent le même objectif mais pour répondre à des besoins divers. Enfin, nous étudions la problématique de la conception du schéma d'un entrepôt. Une fois ces thématiques présentées, nous discutons de leurs inadéquations vis-à-vis de notre problématique.

3.1 Panorama des types de hiérarchies

Nous dressons maintenant un panorama des différents types de hiérarchies proposés dans la littérature et nous appuyons pour cela sur la catégorisation proposée dans [MZ04].

Les hiérarchies simples

Les hiérarchies simples sont des hiérarchies où les différents niveaux les composant peuvent être considérés comme des listes et leurs instances forment un arbre. Ce sont les hiérarchies que nous avons considérées jusqu'ici dans ce manuscrit. Ces hiérarchies peuvent être symétriques selon que l'arbre des instances est équilibré ou asymétrique s'il ne l'est pas. Par exemple, la hiérarchie *Lieu* est une hiérarchie symétrique alors que la hiérarchie *Vecteur de communication* est asymétrique.

Les hiérarchies non-strictes

Dans ce panorama des catégories de hiérarchies existantes, nous avons supposé jusqu'à présent que la généralisation d'une valeur à un niveau donné ne pouvait conduire qu'à, au plus, une valeur. Par exemple, la généralisation au niveau *Mois* de la date 01/02/2011 est *Février*. En pratique, il existe de nombreuses situations où ce type de hiérarchie est trop restrictif. Typiquement, comme

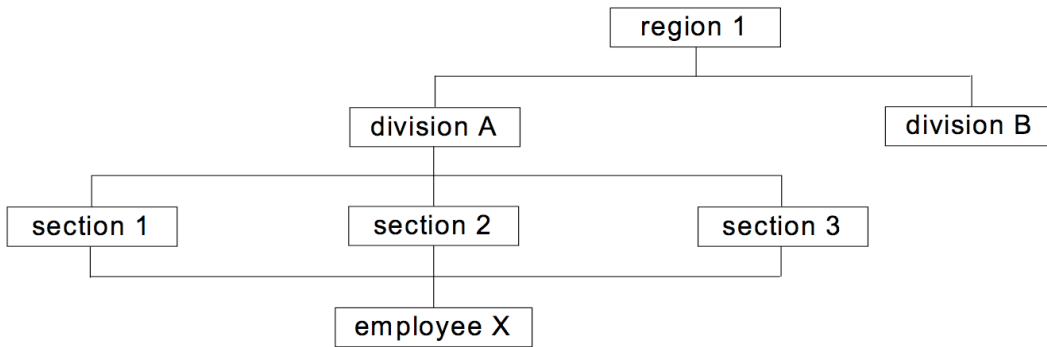


FIGURE 6.1 – Exemple de hiérarchie non stricte (instances)[MZ04]

le montre la figure 6.1, un employé peut appartenir à plusieurs services au sein d’une entreprise. De telles hiérarchies sont dites *non strictes*. Sinon, la hiérarchie est dite *stricte*.

Les hiérarchies floues

Les hiérarchies floues sont un cas particulier de hiérarchies non strictes. En effet, elles permettent de définir des hiérarchies où une instance peut appartenir graduellement à une instance d’un niveau supérieur. Par exemple, l’âge d’une personne peut être rattaché selon un certain degré d’appartenance à la catégorie *Jeune* et selon un autre degré à la catégorie *Adulte*. Comme cela est décrit dans [Lau03], ces hiérarchies floues peuvent être définies grâce à deux cadres formels : les graphes flous ou les partitions floues. La figure 6.2, extraite de [AP03], représente un exemple de hiérarchie floue décrivant le fait que la gamme de gris est une combinaison à des degrés différents de blanc et de noir.

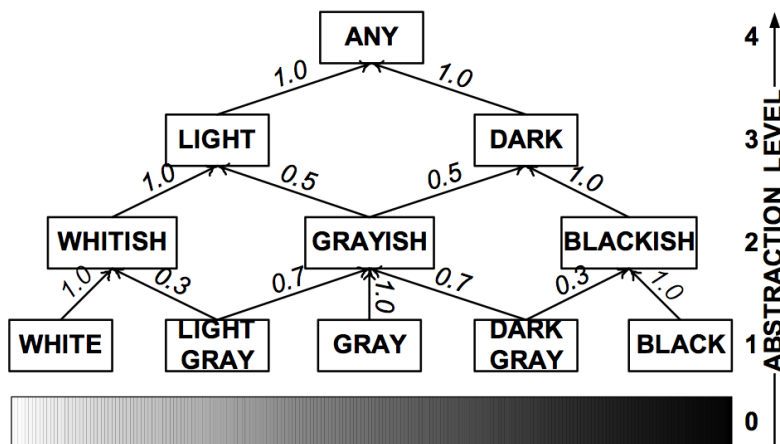


FIGURE 6.2 – Exemple de hiérarchie floue (niveau instances)[AP03]

Les hiérarchies multiples

Les hiérarchies multiples permettent de modéliser les situations où les différents niveaux composant ces hiérarchies ne sont plus des listes mais des graphes orientés sans cycle. Un exemple bien connu de hiérarchie multiple est celui de la hiérarchie associée à la dimension temporelle. La figure 6.3 représente les différents niveaux de granularité pouvant exister dans une dimension temporelle. Nous remarquons qu’il existe deux façons de généraliser le niveau *journée* (*i.e.* sur le niveau *semaine* ou le niveau *mois*) mais que ces deux niveaux ne sont pas comparables (*i.e.* l’un n’est pas une généralisation de l’autre).

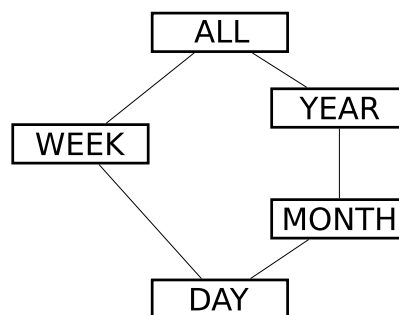


FIGURE 6.3 – Les niveaux associés à une dimension temporelle[MZ04]

Les hiérarchies parallèles

Jusqu’ici, nous avons supposé dans ce manuscrit qu’une dimension n’était composée que d’un attribut et que par conséquent, il ne pouvait exister qu’une hiérarchie par dimension. En pratique, il est fréquent qu’une même dimension soit composée de plusieurs attributs utiles pour l’analyse. Si ces attributs possèdent une hiérarchie associée, la hiérarchie de la dimension est dite *parallèle* et peut être considérée comme un cas particulier des *hiérarchies multiples*. Une hiérarchie parallèle peut être composée de hiérarchies précédemment décrites. On distingue généralement deux catégories de *hiérarchies parallèles*. Une *hiérarchie parallèle indépendante* est telle qu’il n’existe pas de niveau commun entre les différentes hiérarchies la composant. Sinon, la *hiérarchie parallèle* est dite *dépendante*. La figure 6.4, extraite de [MZ04], représente une hiérarchie parallèle indépendante décrivant une dimension *Store* décrite par quatre attributs dont deux possèdent une hiérarchie (*i.e.* l’attribut *Store address* permet la définition d’une hiérarchie géographique et l’attribut *Store number* permet la définition d’une hiérarchie associée à l’organisation structurelle du groupe).

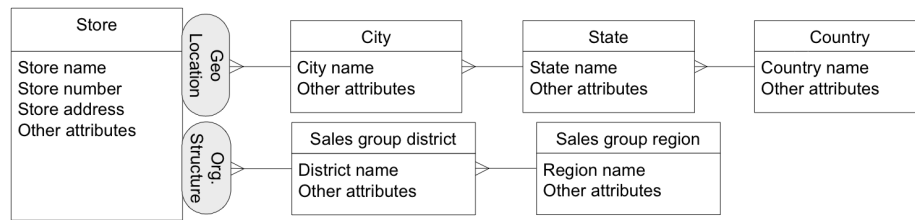


FIGURE 6.4 – Exemple de dimension parallèle indépendante associée à une dimension *Store* [MZ04]

3.2 Panorama des modèles conceptuels d'entrepôts de données

Les concepts et les formalismes associés à la modélisation de bases de données multidimensionnelles existent mais souffrent de l'absence d'un consensus standardisé [RALT06]. Malgré tout, les concepts reposant sur la métaphore du cube [GCB⁺97] et présentés dans la section 1.1 du chapitre 2 (page 23) sont présents dans la plupart des modélisations existantes. Pour rappel, la modélisation multidimensionnelle consiste à modéliser les sujets d'analyse appelés *faits* autour d'axes d'analyse appelés *dimensions*. Les faits sont constitués d'indicateurs d'analyse appelés *mesures* qui correspondent aux données des cellules du cube. Les dimensions sont composées d'attributs modélisant les différents niveaux de granularité des axes d'analyse. Ces attributs composent ainsi la hiérarchie de la dimension à laquelle ils sont attachés. Chaque dimension représente une arête du cube.

Face au succès des entrepôts de données dans le monde de l'entreprise, le besoin de proposer des modèles conceptuels simples, compréhensibles par des non informaticiens fut nécessaire. Ainsi, en l'absence de standardisation, différents modèles et leur formalisme graphique furent proposés. Les premières modélisations multidimensionnelles s'inspiraient directement de la métaphore du cube [GCB⁺97, AGS97]. Ces approches souffrent de quelques limites que nous évoquons rapidement ici. Tout d'abord, il n'est pas aisé de représenter les différentes hiérarchies des données. Ensuite, la représentation graphique de ces modèles sous forme de cube ne permet pas de considérer les cas où plus de trois dimensions d'analyse existent. Enfin, ce modèle peine à représenter les cas, très courants, où plusieurs faits existent dans l'entrepôt et que ces faits partagent des dimensions d'analyse communes. Pour pallier ces limites, de nouveaux modèles ont été proposés et peuvent globalement se diviser en deux catégories : les modèles étendant des paradigmes existants ou les modèles totalement originaux. La première catégorie étend des paradigmes tels que le modèle entité/relation [SBHD04, MZ06] ou s'appuient sur le paradigme objet via un formalisme UML [PJ99, ASS06]. En étendant des paradigmes existants, ces approches introduisent souvent de nombreux concepts supplémentaires qui gênent la compréhension du modèle. La deuxième catégorie de modèles, dit *multidimensionnels* [CT98, GMR98b, RTZ01, RTT08], ne repose sur aucun paradigme existant et développe un nombre très faible de concepts (fait, dimension, hiérarchie). Cependant, malgré

ce faible nombre de concepts manipulés par ces approches, ces modèles représentent la structure des dimensions de manières très différentes [Tor03].

Indépendamment de la modélisation adoptée, un fait et ses dimensions associées composent un schéma en étoile [Kim96]. Ce modèle a été généralisé pour permettre la description d'une *constellation* d'étoiles tels que plusieurs faits sont présents dans le modèle et où plusieurs dimensions (éventuellement partagées) forment un schéma en *constellation* [Kim96]. Récemment, un nouveau modèle multidimensionnel, dit *en galaxie* [Tou07], fut proposé pour modéliser des entrepôts de documents et s'abstraire de la distinction entre *fait* et *dimension*.

Exemple 6.1 La figure 6.5 représente l'entrepôt classique présenté dans le cas d'étude de ce chapitre. Par classique, nous entendons le fait que les particularités inhérentes à notre problématique (i.e. hiérarchie sur mesure, contextualisation du lien d'agrégation) ne sont pas représentées. Le formalisme introduit dans [RTTZ08] et inspiré de [GMR98b] est utilisé. Ce modèle en constellation comprend deux faits symbolisés par un rectangle vert (i.e. Tension et Posologie) et trois dimensions symbolisées par un rectangle rouge (i.e. Patient, Temps, Médicaments) dont deux sont communes aux deux faits représentés dans l'entrepôt. Plusieurs hiérarchies existent dans cet entrepôt. Parmi elles, nous citons la hiérarchie $Geo = idPatient, Ville, Pays, ALL_{Patient}$ associée à la dimension Patient. Enfin, un attribut est considéré comme faible, s'il ne représente pas un axe d'analyse. Par exemple, l'attribut Nom de la dimension Patient est un attribut faible.

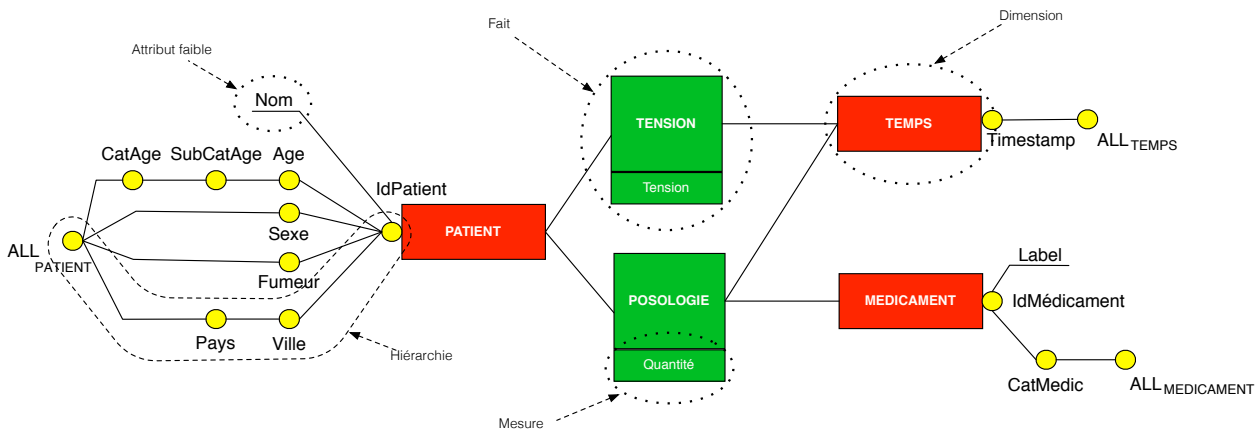


FIGURE 6.5 – Schéma classique de l'entrepôt de données associé au cas d'étude de ce chapitre

3.3 Panorama des approches introduisant de la flexibilité dans l'analyse des données

Compte tenu de notre problématique de hiérarchisation contextuelle d'attributs (non figée vis-à-vis du chemin d'agrégation), nous nous sommes intéressés aux différents travaux introduisant une certaine forme de flexibilité ou de capacité d'expression au niveau des dimensions, des hiérarchies.

Afin de pouvoir rendre l'analyse plus flexible, un langage à base de règles a été développé dans [EV01] pour la gestion des exceptions dans le processus d'agrégation. Le langage IRAH (Intentional Redefinition of Aggregation Hierarchies) permet de redéfinir des chemins d'agrégation pour exprimer des exceptions dans les hiérarchies de dimensions prévues par le modèle. Ce travail permet aux utilisateurs d'exprimer eux-mêmes les exceptions dans le processus d'agrégation. En effet, afin de prendre en compte ces exceptions, les utilisateurs définissent et exécutent un programme IRAH, produisant ainsi une révision des chemins d'agrégation. L'exemple considéré est l'étude des prêts d'une compagnie de crédit en fonction de la dimension emprunteur qui est hiérarchisée. La catégorie de l'emprunteur est définie en fonction de son revenu. Mais les auteurs expliquent qu'il est possible que l'analyste veuille ré-affecter un emprunteur dans une autre catégorie en voulant tenir compte d'autres paramètres que le revenu. Dans ce cas, le processus d'agrégation doit tenir compte de cette «exception». Dans ces travaux les auteurs proposent alors un langage à base de règles qui permet de définir des analyses révisées qui tiennent compte de ce type d'exception.

Si ce langage constitue une alternative à la rigidité du schéma multidimensionnel dans le processus d'agrégation pour les utilisateurs, il ne fait qu'en modifier les chemins en fonction d'exceptions dans les hiérarchies de dimension. Or dans notre cas, il ne s'agit pas de prendre en compte des exceptions, mais bel et bien de prendre en compte des chemins d'agrégation qui dépendent d'un contexte, au niveau des mesures, en se basant sur des connaissances d'experts.

Dans [FBB07], un précédent travail s'est penché sur la proposition d'un modèle d'entrepôt à base de règles qui visait à représenter la création de nouveaux niveaux d'analyse pour répondre à un besoin de personnalisation des analyses en fonction de la connaissance individuelle d'utilisateurs. Les nouveaux niveaux étaient créés le long des hiérarchies de dimension (insertion d'un niveau ou ajout en fin de hiérarchie), forcément au-dessus du premier niveau (dimension). Le lien d'agrégation exprimé ne peut être directement lié à la table des faits, empêchant l'expression d'une hiérarchisation de mesure comme nous avons besoin de le faire.

Une alternative pouvant présenter un certain intérêt vis-à-vis du problème posé par la nécessité de prendre en compte le fait que les chemins d'agrégation le long d'une hiérarchie peuvent changer est le versionnement. Il s'agit de pouvoir exprimer un changement (plutôt temporel) au niveau des instances de dimension comme proposé par [BSSJ98], au niveau des liens d'agrégation comme envisagé par [MV00] ou de la structure comme proposé par [MW03]. Ceci répond au problème

évoqué dans l'introduction sur l'indépendance des dimensions, par rapport à la dimension temporelle entre autres. Le versionnement est mis en avant pour la possibilité de stocker le fait que les dimensions puissent évoluer. Le versionnement permet donc non seulement d'historiser les données de la table des faits à travers une dimension temporelle, mais également les modifications au sein même des hiérarchies de dimension. Le problème majeur de ces approches est que ces versions sont développées par rapport à une évolution dans le temps et ne sont pas faites pour prendre en compte une modification non pas dans le temps mais par rapport à ce que nous avons appelé «contexte». Par ailleurs, cela ne résout pas le problème du point de vue de la hiérarchisation de mesure que nous avons besoin de modéliser.

3.4 Panorama des méthodes de conception du schéma d'un entrepôt

Du point de vue de la conception du schéma de l'entrepôt, nous distinguons dans la littérature trois grandes approches : celle guidée par les données, qualifiée également d'ascendante ; celle guidée par les besoins d'analyse, dénommée également descendante et l'approche mixte qui combine les deux précédentes [SFG05].

L'approche orientée données ignore les besoins d'analyse a priori. Elle concerne en particulier les travaux sur l'automatisation de la conception de schéma. En effet, cette approche consiste à construire le schéma de l'entrepôt à partir de ceux des sources de données et suppose que le schéma qui sera construit pourra répondre à tous les besoins d'analyse. Par exemple, [GMR98a] proposent une méthodologie semi-automatique pour construire un schéma d'entrepôt de données à partir des schémas entité-relation qui représentent les bases de données sources. [PIR03] représentent les connaissances sur la construction de l'entrepôt de données sous forme de règles. Un algorithme gère alors l'ordre d'exécution des règles qui permettent une succession de transformations sur le schéma source pour obtenir le schéma logique final de l'entrepôt.

Les approches orientées besoins d'analyse, quant à elles, proposent de définir le schéma de l'entrepôt en fonction des besoins d'analyse et supposent que les données disponibles permettront la mise en œuvre d'un tel schéma. Parmi les approches orientées besoins d'analyse, [LBMS02] proposent une distinction entre les approches guidées par les buts et les approches guidées par les utilisateurs.

L'approche orientée buts suppose que le schéma de l'entrepôt est défini selon les objectifs d'analyse de l'entreprise [BvE00]. Ainsi, on suppose que tous les employés de l'entreprise ont des besoins d'analyses similaires vis-à-vis de l'exploitation de l'entrepôt de données. Autrement dit, tous les employés ont la même vision analytique de l'entrepôt de données.

Dans l'approche orientée utilisateurs, ces derniers sont interrogés afin de collecter l'ensemble de leurs besoins d'analyse avant de construire l'entrepôt de données, ce qui permet de garantir

l'acceptation du système par les utilisateurs. Cependant la durée de vie de ce schéma peut être courte, étant donné que le schéma dépend beaucoup des besoins des personnes impliquées dans le processus de développement de l'entrepôt. Pour y remédier, dans [Poe96], l'auteur propose une méthodologie pour conduire les entretiens avec les utilisateurs pour la collecte des besoins. Il est alors recommandé d'interroger différents groupes d'utilisateurs pour avoir la vision la plus complète possible des besoins des utilisateurs. Mais reconnaissons qu'il est non seulement difficile de déterminer de façon exhaustive les besoins d'analyse pour l'ensemble des utilisateurs à un instant donné, mais qu'il est encore moins facile de déterminer leurs besoins à venir.

Dans ces deux approches, nous considérons tout simplement qu'il s'agit de besoins d'analyse qui sont exprimés, certains sont plus globaux, au sens où l'ensemble des utilisateurs partage ce besoin, d'autres plus spécifiques. Enfin, l'approche mixte considère à la fois les besoins d'analyse et les données pour la construction du schéma. Cette approche est celle qui fait l'objet de plus d'investigations aujourd'hui. L'idée générale est de construire des schémas candidats à partir des données (démarche ascendante) et de les confronter aux schémas définis selon les besoins (démarche descendante) [BCC⁺01, PD02, SFG05]. Ainsi, le schéma construit constitue une réponse aux besoins réels d'analyse et il est également possible de le mettre en œuvre avec les sources de données. Il apparaît donc important dans le cadre de cette démarche de pouvoir discuter des schémas compréhensibles avec les utilisateurs (experts du domaine).

3.5 Discussion

Premièrement, nous relevons qu'aucune catégorie de hiérarchies présentée dans la section 3.1 ne répond parfaitement à nos besoins. En effet, une hiérarchie intégrant les contextes de généralisations implique que pour un élément, sa généralisation contextuelle est unique (*i.e.* hiérarchie stricte) mais peut différer en fonction du contexte. Par exemple, la généralisation au niveau *Cat-Tension* d'une tension artérielle ne sera pas la même en fonction des patients considérés. Ainsi, à l'échelle de l'individu, une hiérarchie intégrant les contextes est considérée comme stricte mais, d'un point de vue plus global, une même valeur possède plusieurs généralisations sur le même niveau (*i.e.* hiérarchie non-stricte). Les hiérarchies floues ne peuvent pas non plus répondre à nos besoins car (1) nous souhaitons une hiérarchie stricte à l'échelle d'un individu et (2) le domaine de définition des fonctions d'appartenance aux sous-ensembles flous (*e.g.* faible, normal, élevé) n'est composé que des valeurs à généraliser. Par exemple, une tension à 10 sera généralisée à faible à 90% et normale à 10% sans considérer les caractéristiques du patient associé à cette tension. Dans ce travail, nous nous plaçons dans un contexte de hiérarchies simples (*i.e.* les niveaux d'agrégation représentent un chemin) et parallèles dépendantes (*i.e.* il peut exister plusieurs hiérarchies associées à une même dimension et ces hiérarchies partagent deux niveaux communs : le plus spécifique

et le plus général).

Ensuite, puisque nous venons de voir qu’aucun type de hiérarchies présent dans la littérature ne correspond à nos attentes, les modélisations conceptuelles d’entrepôts de données existantes n’intègrent naturellement pas ce type de hiérarchie. Dans ce travail, la modélisation multidimensionnelle fut retenue à cause de la simplicité des concepts associés. Parmi les modèles multidimensionnels existants, nous nous sommes inspirés du modèle proposé dans [RTTZ08] (lui même inspiré de [GMR98b]). Malgré les possibilités intéressantes associées à ce modèle, deux raisons expliquent son inadéquation dans la problématique étudiée ici : l’impossibilité de définir une hiérarchie sur une mesure et l’impossibilité d’indiquer qu’une généralisation est contextualisée. Pour s’en convaincre, considérons la figure 6.5. Premièrement, la définition d’une hiérarchie $Tension = (Tension, CatTension, NormalitéTension)$ est impossible car, par définition, un fait n’est composé que des identifiants des dimensions associées et de mesures. Ici, cela impliquerait que (1) la mesure *Tension* soit considérée comme un attribut hiérarchisé et (2) que de nouveaux attributs hiérarchisés (*i.e.* *CatTension* et *NormalitéTension*) soit inclus dans le fait *Tension*. En outre, comme nous pouvons le remarquer sur la figure 6.5, la généralisation à un certain niveau n’est guidée que par le niveau inférieur (représenté graphiquement par le lien entre ces deux niveaux). Dès lors, il est impossible de modéliser le fait que, par exemple, la généralisation au niveau *SubCatAge* est conditionnée aussi bien par *l’âge* que par l’attribut *Fumeur*. Ces deux limites nous ont conduits à la proposition d’un nouveau modèle multidimensionnel que nous présentons dans la suite de ce chapitre.

Enfin, si l’ensemble des travaux présentés dans la section 3.3 apportent une flexibilité dans l’analyse des données en se focalisant sur ce qui définit en l’occurrence le contexte d’analyse, à savoir les dimensions, ils ne prennent pas en compte le besoin de flexibilité au niveau de la mesure. En outre, il ne s’agit pas de considérer des hiérarchies dépendant uniquement de la valeur de la mesure. A notre connaissance, il n’y a pas de travaux proposant ou permettant de solutionner ce problème.

4 Modélisation conceptuelle et graphique

Dans cette section, nous présentons un modèle conceptuel multidimensionnel permettant la prise en compte des hiérarchies contextuelles définies indifféremment sur des attributs ou des mesures. De plus, devant le succès grandissant des entrepôts de données dans le secteur industriel, il est important de fournir des outils de conception efficaces et complets aux futurs utilisateurs du système. Dès lors, un formalisme graphique permettant une visualisation simple et facilement

interprétable de la structure de l'entrepôt de données apparaît être un besoin pertinent. Aussi, nous associons au modèle conceptuel multidimensionnel présenté, une formalisation graphique de ce modèle.

Nous notons $\mathcal{A} = \{A_1, \dots, A_t\}$ l'ensemble des t attributs de dimensions et $\mathcal{M} = \{M_1, \dots, M_d\}$ l'ensemble des k mesures de l'entrepôt. Nous désignons par Ω l'union des attributs et des mesures (*i.e.* $\Omega = \mathcal{A} \cup \mathcal{M}$). Dans la suite de ce chapitre, le terme générique *attribut* désignera indifféremment un attribut de dimension ou une mesure.

Définition 6.1 (*Chemin de généralisation*) Soient X et Y deux éléments de Ω et \mathcal{C} un sous-ensemble de Ω . Un chemin de généralisation, noté G , entre X et Y , est défini par $G = X \xrightarrow{\mathcal{C}} Y$ tel que :

1. $X \in \mathcal{C}$
2. $Y \notin \mathcal{C}$
3. Si $X \in \mathcal{A}$ (resp. $X \in \mathcal{M}$) alors $Y \in \mathcal{A}$ (resp. $Y \in \mathcal{M}$)
4. $\mathcal{C} - X \subseteq \mathcal{A}$
5. Y dépend fonctionnellement de \mathcal{C} .

Définition 6.2 (*Chemin de généralisation contextuel et classique*) Soit $G = X \xrightarrow{\mathcal{C}} Y$ un chemin de généralisation. G est appelé un chemin de généralisation contextuel si $|\mathcal{C}| > 1$. Sinon, (*i.e.*, $\mathcal{C} = \{X\}$), G est appelé un chemin de généralisation classique (ou simplement un chemin de généralisation quand cela est sans ambiguïté).

Exemple 6.2 D'après le cas d'étude, le chemin de généralisation $G_1 = IdPatient \xrightarrow{\mathcal{C}_1} Sexe$, avec $\mathcal{C}_1 = \{IdPatient\}$ est un chemin de généralisation classique. Par contre, $G_2 = Age \xrightarrow{\mathcal{C}_2} SubCatAge$, avec $\mathcal{C}_2 = \{Age, Fumeur\}$ est un chemin de généralisation contextuel.

Définition 6.3 (*Attribut contextualisant et contextualisé*) Soit $G = X \xrightarrow{\mathcal{C}} Y$ un chemin de généralisation contextuel. L'attribut Y est dit contextualisé par les attributs de \mathcal{C} . Les attributs de \mathcal{C} sont appelés attributs contextualisants.

Si Y est une mesure (resp. un attribut de dimension), *i.e.* $Y \in \mathcal{M}$ (resp. *i.e.* $Y \in \mathcal{A}$), Y est appelée *mesure contextualisée* (resp. *attribut de dimension contextualisé*). Similairement, les attributs contextualisés de \mathcal{C} sont appelés attributs de dimension ou mesures contextualisants selon leur

nature (*i.e.* attribut ou mesure). Il est important de préciser qu'un même attribut peut être contextualisé dans un contexte et contextualisant dans un autre. Par exemple, l'attribut *SubCatAge* est contextualisant car sa valeur impacte sur la généralisation de la tension mais est contextualisé puisque généraliser à ce niveau implique de considérer aussi bien l'âge du patient que la valeur de l'attribut *Fumeur*.

Définition 6.4 (*Représentation des chemins de généralisation*) Soit $G = X \xrightarrow{\mathcal{C}} Y$ avec $\mathcal{C} = \{C_1, \dots, C_k\}$ un chemin de généralisation. La représentation graphique associée à G sera différente en fonction de sa nature :

- Si G est un chemin classique entre attributs de dimension, le formalisme graphique associé est celui présenté dans la figure 6.6(a). Les attributs de dimension sont représentés par des petits cercles jaunes reliés entre eux par un trait plein);
- Si G est un chemin classique entre mesures, le formalisme graphique associé est celui présenté dans la figure 6.6(b). Les mesures sont représentées par des petits cercles verts reliés entre eux par un trait plein);
- Si G est un chemin contextuel entre attributs de dimension, le formalisme graphique associé est celui présenté dans la figure 6.6(c). L'attribut de dimension X est représenté par un cercle jaune et l'attribut contextualisé Y par un satellite entourant un cercle jaune. La liste des éléments contextualisants (*i.e.* les membres de \mathcal{C}) est adossée au satellite et le lien entre X et Y est représenté par un trait plein;
- Si G est un chemin contextuel entre mesures, le formalisme graphique associé est celui présenté dans la figure 6.6(d). La mesure X est représentée par un cercle vert et la mesure contextualisée Y par un satellite entourant un cercle vert. La liste des éléments contextualisants (*i.e.* les membres de \mathcal{C}) est adossée au satellite et le lien entre X et Y est représenté par un trait plein.

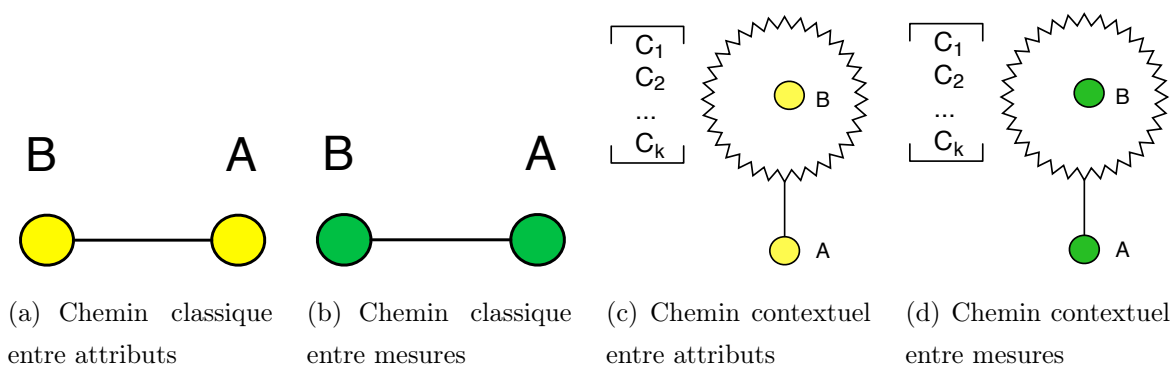


FIGURE 6.6 – Représentation graphique des chemins de généralisation

Définition 6.5 (*Contexte : structure*) Soit $G = X \xrightarrow{\mathcal{C}} Y$ un chemin de généralisation contextuel. Un contexte, ϕ , est un triplet $\phi = (IdC, \mathcal{C}, Y)$ où IdC est l'identifiant du contexte et Y est un attribut contextualisé par \mathcal{C} . Nous notons $\Phi = \{\phi_1, \dots, \phi_v\}$ l'ensemble des contextes de l'entrepôt.

Exemple 6.3 Dans l'entrepôt médical considéré dans ce chapitre, un contexte possible est $\phi_1 = (Tension, \{Tension, SubCatAge, Fumeur\}, CatTension)$.

Dans la suite, nous désignerons un contexte par son identifiant plutôt que par sa notation ϕ_i et par convention, nous le désignons par l'attribut contextualisé (e.g. l'identifiant du contexte associé à la généralisation d'une tension mesurée est *Tension*).

Définition 6.6 (*Contexte : instances*) Soit $G_i = X_i \xrightarrow{\mathcal{C}_i} Y_i$ un chemin de généralisation contextuel et $\phi_i = (IdC_i, \mathcal{C}_i, Y_i)$ (avec $\mathcal{C}_i = \{C_{i1}, \dots, C_{ik}\}$) le contexte associé. Une instance de contexte correspond à l'instanciation des éléments de ϕ_i . Formellement, une instance ϕ_{ij} du contexte ϕ_i est de la forme $\phi_{ij} = (IdC_i, IdInst_{ij}, \Xi_{ij}, y_{ij})$ tel que :

- IdC_i désigne l'identifiant du contexte ϕ_i ;
- $IdInst_{ij}$ désigne l'identifiant de l'instance de contexte ϕ_{ij} ;
- Ξ_{ij} est un ensemble de conditions de la forme $\Xi_{ij} = \{(C_{i1}, Cond_{i1}), \dots, (C_{ik}, Cond_{ik})\}$ où $Cond_{il} \subseteq Dom(C_{il})$. $Cond_{il}$ peut être défini en extension ou en compréhension ;
- y_{ij} désigne la généralisation de x_{ij} ($x_{ij} \in Dom(X_i)$) si toutes les conditions de Ξ_{ij} sont respectées.

La sémantique associée à ϕ_{ij} est :

$$\text{SI } c_{i1_j} \in Cond_{i1_j} \wedge \dots \wedge c_{ik_j} \in Cond_{ik_j} \text{ ALORS } Y_i = y_{ij} \text{ (avec } c_{il_j} \in Dom(C_{il}) \text{)}$$

Exemple 6.4 *Instanciation d'attribut contextualisé* : Chaque ligne du tableau 6.1 représente une instance du contexte *Tension*. Ainsi, selon notre formalisme, la première ligne peut être retranscrite ainsi : $\phi_{11} = (Tension, 1, \{(SubCatAge, \{Nourrisson\}), (Fumeur, \{Oui, Non\}), (Tension, \{x|x > 12\})\}, Elevée)$.

Définition 6.7 (*Hiérarchie*) Soit $H_i = (X_{i1}, \dots, X_{imax_i})$ une séquence d'attributs de Ω . H_i est une hiérarchie s'il existe un chemin de généralisation entre chaque paire adjacente de H_i . Formellement, H_i est une hiérarchie si $X_{i1} \xrightarrow{\mathcal{C}_{i2}} X_{i2} \dots X_{i(max_i-1)} \xrightarrow{\mathcal{C}_{imax_i}} X_{imax_i}$.

Remarquons que la Définition 6.1 implique que si X_{i1} est une mesure (resp. un attribut de dimension), tous les membres de H_i sont des mesures (resp. des attributs de dimension). Dans ce cas, H_i est appelée *hiérarchie de mesures* (resp. *hiérarchie d'attributs de dimension*).

Définition 6.8 (*Hiérarchie contextuelle et classique*) Soit $H_i = (X_{i1}, \dots, X_{imax_i})$ une hiérarchie. H_i est dite contextuelle si au moins un chemin de généralisation entre deux niveaux de H_i est contextuel. Sinon, H_i est dite classique.

Une hiérarchie étant une succession de chemins de généralisation, sa représentation graphique découle directement des représentations présentées dans la Définition 6.4.

Définition 6.9 (*Dimension*) Une dimension D_i est définie comme $D_i = (DN_i, Id_i, \mathcal{H}_i, \mathcal{AF}_i)$ où DN_i désigne le nom donné à D_i , Id_i désigne un attribut, \mathcal{H}_i désigne un ensemble de hiérarchies d'attributs de dimensions et \mathcal{AF}_i désigne un ensemble d'attributs de dimensions avec :

1. Id_i désigne l'identifiant de D_i (i.e. détermine fonctionnellement tous les attributs associés à D_i);
2. L'attribut débutant toute hiérarchie H_{il} de \mathcal{H}_i est PK_i (i.e. $\forall H_{il} = (X_{il1}, \dots, X_{ilmax_{il}})$ où $H_{il} \in \mathcal{H}_i$, $X_{il1} = PK_i$);
3. L'attribut terminant toute hiérarchie H_{il} de \mathcal{H}_i est une valeur joker, ALL_i , désignant toutes les instances de D_i (i.e. $\forall H_{il} = (X_{il1}, \dots, X_{ilmax_{il}})$ où $H_{il} \in \mathcal{H}_i$, $X_{ilmax_{il}} = ALL_i$);
4. Mis à part les premiers et derniers attributs de chaque hiérarchie, un attribut ne peut apparaître dans deux hiérarchies de \mathcal{H}_i (i.e. $\forall X_{ijk}, X_{ilm}$ tels que $X_{ijk} \in H_{ij} - \{X_{ij1}, X_{ijmax_{ij}}\}$ et $X_{ilm} \in H_{il} - \{X_{il1}, X_{ilmax_{il}}\}$, $X_{ijk} \neq X_{ilm}$);
5. $\mathcal{AF}_i = \{AF_{i1}, \dots, AF_{it}\}$ désigne un ensemble d'attributs dits faibles rattachés au niveau le plus précis.

Exemple 6.5 Selon le formalisme introduit, la dimension Patient est définie ainsi : $D_1 = (Patient, IdPatient, \{(IdPatient, Age, SubCatAge, CatAge, ALL_{Patient}), (IdPatient, Sexe, ALL_{Patient}), (IdPatient, Fumeur, ALL_{Patient}), (IdPatient, Ville, Pays, ALL_{Patient})\}, \{Nom\})$.

Définition 6.10 (*Représentation des dimensions*) Une dimension $D_i = (DN_i, Id_i, \mathcal{H}_i, \mathcal{AF}_i)$ avec $\mathcal{H}_i = \{H_{i1}, \dots, H_{ik}\}$ et $\mathcal{AF}_i = \{AF_{i1}, \dots, AF_{il}\}$ est représentée grâce au formalisme

graphique présenté dans la figure 6.7.

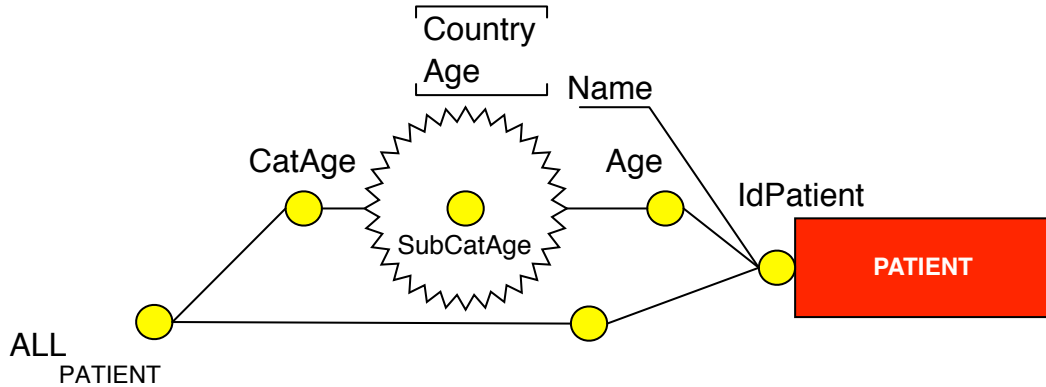


FIGURE 6.7 – Représentation graphique d'une dimension D_i

Nous notons $\mathcal{D} = D_1, \dots, D_n$ l'ensemble des n dimensions de l'entrepôt. Pour une dimension D_i , nous notons \mathcal{A}_i l'ensemble des attributs qui sont associés à D_i (l'identifiant, les attributs compris dans les différentes hiérarchies et les attributs faibles). Dans le modèle proposé, un attribut ne peut appartenir à plusieurs dimensions et l'union des attributs des dimensions représente l'ensemble des attributs présents dans l'entrepôt. Nous avons donc $\bigcup_{1 \leq i \leq n} \mathcal{A}_i = \mathcal{A}$ et $\bigcap_{1 \leq i \leq n} \mathcal{A}_i = \emptyset$.

Définition 6.11 (Instance de dimension) Soit $D_i = (DN_i, Id_i, \mathcal{H}_i, \mathcal{AF}_i)$ une dimension. Une instance d_{ij} de la dimension D_i correspond à l'instanciation correcte (par rapport aux hiérarchies) de tous les attributs de D_i excepté ALL_i . Formellement, $d_{ij} = \{(A_{ij}, a_{ijk}) \mid A_{ij} \in (\mathcal{A}_i - ALL_i) \text{ et } a_{ijk} \in \text{Dom}(A_{ij})\}$.

Exemple 6.6 Une instance de la dimension D_2 associée à Médicament est $d_{21} = \{(idMédicament, 1), (Label, Rasilex), (CatMedic, Hypotenseur)\}$ avec Hypotenseur la généralisation de Rasilex.

Définition 6.12 (Fait) Un fait F_i est définie comme $F_i = (FN_i, \mathcal{ID}_i, \mathcal{H}_i, \mathcal{MF}_i)$ où FN_i est le nom donné à F_i , \mathcal{ID}_i désigne un ensemble d'attributs, \mathcal{H}_i désigne un ensemble de hiérarchies de mesures et \mathcal{MF}_i désigne un ensemble d'attributs avec :

1. $\mathcal{ID}_i = \{Id_{i1}, \dots, Id_{il}\}$ détermine fonctionnellement toutes les mesures associées à F_i et chaque Id_{ij} (avec $1 \leq j \leq l$) est un identifiant d'une dimension rattachée à F_i ;

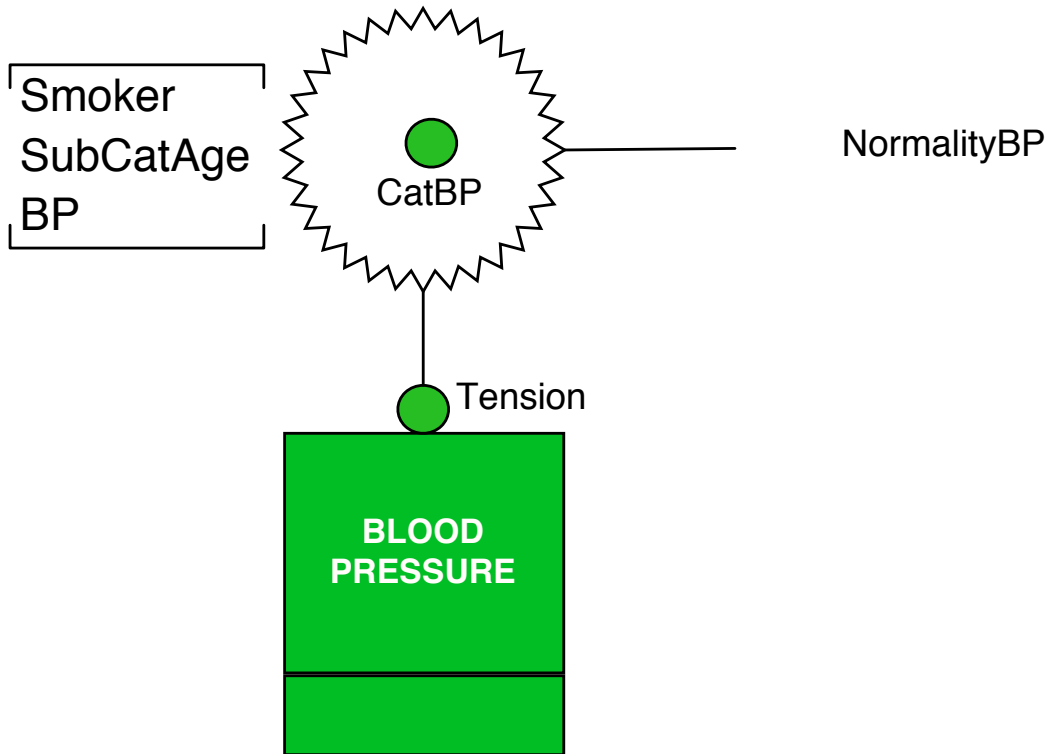


FIGURE 6.8 – Représentation graphique d'un fait F_i

2. Une mesure ne peut apparaître dans deux hiérarchies de \mathcal{H}_i (i.e. $\forall M_{ijk}, M_{ilm}$ avec $M_{ijk} \in H_{ij}$ et $M_{ilm} \in H_{il}$, $M_{ijk} \neq M_{ilm}$);
3. $\mathcal{MF}_i = \{MF_{i1}, \dots, MF_{it}\}$ désigne un ensemble de mesures dites faibles (i.e. n'appartenant à aucune hiérarchie de \mathcal{H}_i).

Exemple 6.7 Selon le formalisme introduit, le fait Tension est défini ainsi : $F_1 = (Tension, \{IdPatient, Timestamp\}, \mathcal{H}_1 = \{(Tension, CatTension, NormalitéTension)\}, \emptyset)$.

Définition 6.13 (Représentation des faits) Un fait $F_i = (FN_i, \mathcal{ID}_i, \mathcal{H}_i, \mathcal{MF}_i)$ avec $\mathcal{ID}_i = \{Id_{i1}, \dots, Id_{im}\}$, $\mathcal{H}_i = \{H_{i1}, \dots, H_{ik}\}$ et $\mathcal{MF}_i = \{MF_{i1}, \dots, MF_{it}\}$ est représentée grâce au formalisme graphique présenté dans la figure 6.8.

Nous notons $\mathcal{F} = F_1, \dots, F_k$ l'ensemble des k dimensions de l'entrepôt. Pour un fait F_i , nous notons \mathcal{M}_i l'ensemble des mesures associées à F_i (les mesures comprises dans les différentes hiérarchies et les mesures faibles). Dans le modèle proposé, une mesure ne peut appartenir à plusieurs

faits et l'union des mesures de chaque fait représente l'ensemble des mesures présentes dans l'entrepôt. Nous avons donc $\bigcup_{1 \leq i \leq k} \mathcal{M}_i = \mathcal{M}$ et $\bigcap_{1 \leq i \leq k} \mathcal{M}_i = \emptyset$.

Définition 6.14 (*Instance de fait*) Soit $F_i = (FN_i, \mathcal{ID}_i, \mathcal{H}_i, \mathcal{MF}_i)$. Une instance f_{ij} du fait F_i correspond à l'instanciation correcte (par rapport aux hiérarchies) de tous les éléments de F_i . Formellement, $f_{ij} = \{(E_{ij}, e_{ijk}) \mid E_{ij} \in (\mathcal{M}_i \cup \mathcal{ID}) \text{ et } e_{ijk} \in \text{Dom}(E_{ij})\}$.

Grâce à ces précédentes définitions, nous pouvons maintenant définir un entrepôt de données contextuel.

Définition 6.15 (*Entrepôt de données contextuel*) Un entrepôt contextuel, CDW , est défini comme un couple d'ensembles de dimensions et de faits, $CDW = (\mathcal{D}, \mathcal{F})$, tels qu'ils ont été précédemment définis.

Exemple 6.8 La figure 6.9 représente l'entrepôt de données décrit dans le cas d'étude selon le formalisme introduit dans cette section. Nous pouvons relever la présence de trois hiérarchies contextuelles. Une d'entre elles, i.e. $H_1 = (IdPatient, Age, SubCatAge, CatAge, ALL_{Patient})$, est une hiérarchie contextuelle d'attributs alors que les deux autres, i.e. $H_2 = (Tension, CatTensionNormalitéTension)$ et $H_3 = (Quantité, CatQuantité, NormalitéQuantité)$, sont des hiérarchies contextuelles de mesures. Un tel formalisme graphique rend immédiate la compréhension de l'entrepôt modélisé et favorise ainsi les interactions entre les concepteurs et les experts.

5 Exploitation des entrepôts contextuels

Maintenant que nous avons défini le modèle conceptuel pour les entrepôts de données, nous nous intéressons à l'exploitation de ces entrepôts. Pour cela, nous fournissons des méthodes pour répondre aux questions duales suivantes :

1. Comment généraliser contextuellement un élément ?
2. Connaissant la valeur d'un élément contextualisé, comment déterminer les valeurs des éléments contextualisants associés (i.e. la spécialisation conceptuelle) ?

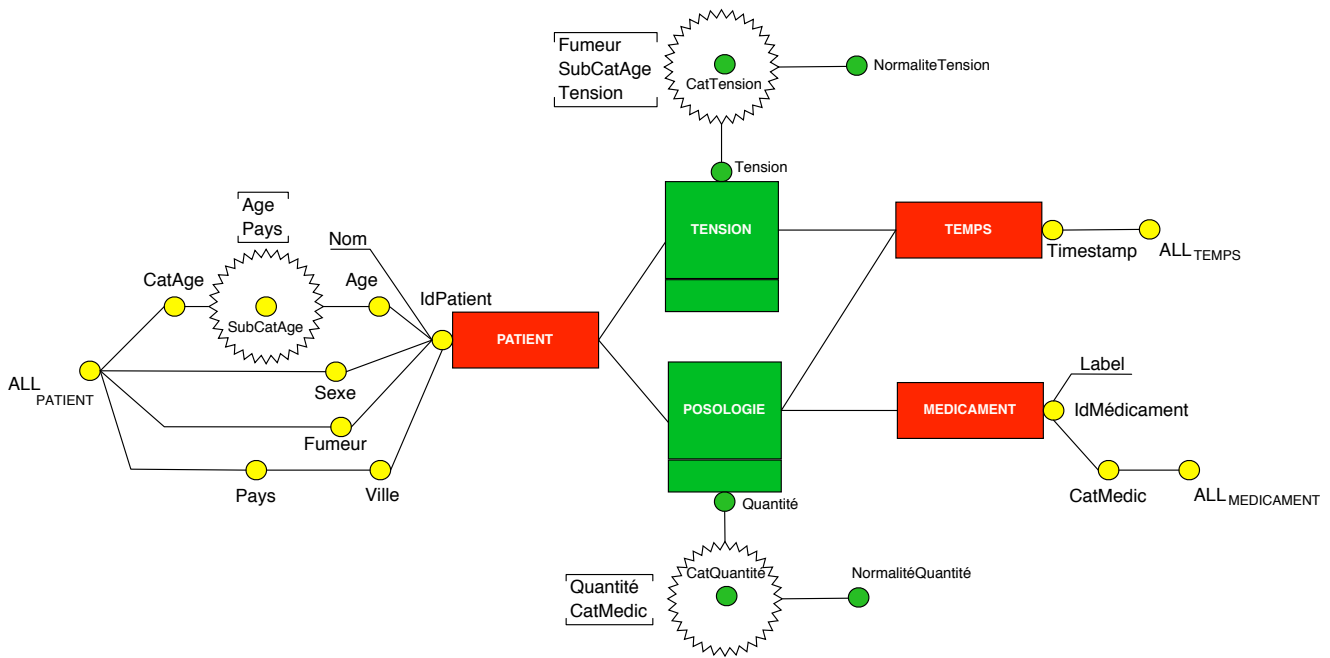


FIGURE 6.9 – Représentation graphique de l'entrepôt de données médicales

Dans un premier temps, nous introduisons quelques définitions préliminaires qui seront utilisées dans la suite de cette section. Nous présentons ensuite les méthodes pour généraliser correctement un élément contextualisé. Enfin, nous décrivons le processus inverse qui consiste à retrouver les instances de contextes associées à partir d'une valeur d'un élément contextualisé.

5.1 Définitions préliminaires

Définition 6.16 (α -extension) Soit id_{ij} une instance de l'identifiant Id_i de la dimension D_i . L' α -extension de id_{ij} est l'instance, d_{ij} , de la dimension D_i telle que α -extension(Id_i, id_{ij}) = d_{ij} avec $(Id_i, id_{ij}) \in d_{ij}$.

Exemple 6.9 Supposons $d_{21} = \{(idMedicament, 1), (Label, Rasilex), (CatMedic, Hypotenseur)\}$ une instance de la dimension $D_2 = Medicament$. Alors, α -extension($idMedicament, 1$) = d_{21} .

Nous définissons maintenant la μ -extension qui peut être considéré comme l'adaptation de l' α -extension aux tables de faits.

Définition 6.17 (μ -extension) Soit $id_{ij} = ((Id_{i1}, id_{i1j}), \dots, (Id_{il}, id_{ilj}))$ une instance de l'identifiant ID_i du fait F_i . La μ -extension de id_{ij} est l'union des α -extension(id_{ikj}) (avec $1 \leq k \leq l$).

Exemple 6.10 Supposons $d_{11} = \{(IdPatient, 1), (Nom, Bob), (Age, 1), (SubCatAge, B  b  ), (CatAge, Jeu)\}$ une instance de la dimension $D_1 = Patient$ et $d_{31} = \{(Timestamp, 01/01/01 - 22 : 22 : 22)\}$ une instance de la dimension $D_3 = Temps$. Alors, μ -extension($(IdPatient, 1), (Timestamp, 01/01/01 - 22 : 22 : 22)$) = $\{(IdPatient, 1), (Nom, Bob), (Age, 1), (SubCatAge, B  b  ), (CatAge, Jeu), (Pays, France), (Timestamp, 01/01/01 - 22 : 22 : 22)\}$.

Nous introduisons maintenant la Π -projection afin de restreindre les fonctions pr  c  demment d  finies    un ensemble d'attributs donn  .

D  finition 6.18 (Π -projection) Soit $\mathcal{E} = \{(E_i, e_{ij}) \mid E_i \in (\mathcal{A} \cup \mathcal{M}) \text{ et } e_{ij} \in Dom(E_i)\}$ un ensemble de couples form  s d'un   l  ment (attribut ou mesure) et d'une valeur appartenant    son domaine de d  finition et soit $\Pi = \{E'_i \mid E'_i \in (\mathcal{A} \cup \mathcal{M})\}$ un ensemble d'attributs. La Π -projection sur l'ensemble \mathcal{E} est l'ensemble des couples (E_i, e_{ij}) de \mathcal{E} tels que $E_i \in \Pi$.

Exemple 6.11 En consid  rant les instances des dimensions utilis  es dans l'exemple pr  c  dent et $\Pi = \{SubCatAge, Fumeur\}$, nous avons Π -projection(μ -extension($(IdPatient, 1), (Timestamp, 01/01/01 - 22 : 22 : 22)$)) = $\{SubCatAge, B  b  ), (Fumeur, Non)\}$.

5.2 G  n  ralisation contextuelle

Une des principales capacit  s offertes par les entrep  ts de donn  es    travers les outils OLAP est la possibilit   de naviguer le long des hi  rarchies des donn  es. D  s lors, nous d  crivons dans cette section comment les contextes et instances de contextes peuvent   tre exploit  s pour r  aliser une g  n  ralisation contextuelle correcte.

G  n  ralisation contextuelle d'attributs de dimension

La d  finition d'une instance de la dimension D_i (D  finition 6.11) suppose que, pour chaque instance id_{ij} de l'identifiant Id_i de D_i , l'ensemble des couples de l' α -extension de id_{ij} soit d  fini. En pratique, la d  finition de l' α -extension est triviale quand D_i ne comporte que des hi  rarchies classiques. Nous supposons en effet que les hi  rarchies classiques sont connues. Par contre, lorsqu'il existe une hi  rarchie contextuelle, le calcul de l' α -extension se r  v  le plus d  licate car il implique de naviguer correctement le long des chemins de g  n  ralisation contextuelle.

Soit $D_i = (DN_i, Id_i, \mathcal{H}_i, \mathcal{AF}_i)$ une dimension avec $\mathcal{H}_i = (H_{i1}, \dots, H_i, \dots, H_{ik})$ l'ensemble des hi  rarchies et soit $id \in Dom(Id_i)$ une instance de l'identifiant Id_i . Nous supposons que

$H = (A_1, \dots, A, B, \dots, A_n)$ est une hiérarchie contextuelle et que $G = A \xrightarrow{\mathcal{C}} B$ est un chemin de généralisation contextuel. Nous supposons de plus le couple (A, a) (tel que $(A, a) \in \alpha\text{-extension}(id)$) connu. Nous recherchons donc à instancier (B, b) (avec $(B, b) \in \alpha\text{-extension}(id)$). Nous adoptons pour cela une méthodologie en quatre étapes :

1. Recherche IdC_i , l'identifiant du contexte $\phi_i = (IdC_i, \mathcal{C}_i, B)$ associé au chemin de généralisation G ;
2. Calculer la Π -projection de l' $\alpha\text{-extension}$ de id avec $\Pi = \mathcal{C}_i$;
3. Rechercher l'instance $\phi_{ij} = (IdC_i, Id_{inst}, \Xi, b)$ du contexte IdC_i qui valide la Π -projection de l' $\alpha\text{-extension}$ de id ;
4. Retourner b .

Généralisation contextuelle de mesures

Similairement à la problématique de généralisation d'attributs que nous venons d'aborder, le calcul de tous les couples membres d'une instance de fait f_{ij} nécessite de calculer correctement une généralisation contextuelle lorsque le fait F_i contient une hiérarchie de mesures contextuelle. Nous décrivons ce processus.

Soit $F_i = (FN_i, \mathcal{ID}_i, \mathcal{H}_i, \mathcal{MF}_i)$ une dimension (avec $\mathcal{H}_i = (H_{i1}, \dots, H, \dots, H_{ik})$), $id_{ij} = (id_{ij1}, \dots, id_{ijl}) \in Dom(\mathcal{ID}_i)$ une instance de l'identifiant \mathcal{ID}_i et f_{ij} l'instance de F_i associée à id_{ij} . Nous supposons que $H = (A_1, \dots, A, B, \dots, A_n)$ est une hiérarchie contextuelle et que $G = A \xrightarrow{\mathcal{C}} B$ est un chemin de généralisation contextuel. Nous supposons de plus le couple (A, a) (tel que $(A, a) \in f_{ij}$) est connu. Nous recherchons donc à instancier (B, b) (avec $(B, b) \in f_{ij}$). Nous adoptons pour cela une méthodologie en quatre étapes :

1. Rechercher IdC_t , l'identifiant du contexte $\phi_i = (IdC_t, \mathcal{C}_t, B)$ associé au chemin de généralisation G ;
2. Calculer la Π -projection de la $\mu\text{-extension}$ de id_{ij} avec $\Pi = \mathcal{C}_i$;
3. Rechercher l'instance $\phi_{tk} = (IdC_t, Id_{insttk}, \Xi_{tk}, b)$ du contexte IdC_t qui valide la Π -projection de la $\mu\text{-extension}$ de id_{ij} ;
4. Retourner b .

5.3 Spécialisation contextuelle

Une des requêtes pouvant être fréquemment formulées par un utilisateur de l'entrepôt de données médicales pourrait être : “*quels sont les patients qui ont eu une tension artérielle très*

faible pendant la nuit ?". Dès lors, répondre à une telle requête implique de disposer d'un moyen de *traduire* qu'est ce qu'une tension basse. Plus généralement, considérons $G = A \xrightarrow{\mathcal{C}} B$ un chemin de généralisation contextuel et $b \in \text{Dom}(B)$. Le processus de spécialisation contextuelle de b peut être décrit en 3 étapes :

1. Rechercher IdC_i , l'identifiant du contexte $\phi_i = (IdC_i, \mathcal{C}_i, B)$ associé au chemin de généralisation G ;
2. Parmi les instances de IdC_i , rechercher celles dont la valeur de l'attribut contextualisé est b . Nous notons $\mathcal{I} = \{\phi_{ij} \mid \phi_{ij} = (IdC_i, Id_{inst_{ij}}, \Xi_{ij}, b_{ij}) \text{ et } b_{ij} = b\}$;
3. Retourner l'ensemble $S = \{\Xi_{ij} \mid \phi_{ij} = (IdC_i, Id_{inst_{ij}}, \Xi_{ij}, b_{ij}) \text{ et } \phi_{ij} \in \mathcal{I}\}$ tel que chaque Ξ_{ij} représente une conjonction de critères à valider pour obtenir b (*i.e.* S représente une disjonction de conjonction de critères conduisant à l'instance b de l'élément contextualisé B).

6 Mise en œuvre

6.1 Stockage des connaissances : utilisation d'une base de connaissance externe

Cette section aborde la problématique de la représentation et du stockage de ces connaissances expertes dans l'hypothèse d'une mise en œuvre dans un système relationnel (ROLAP) pour le stockage de l'entrepôt de données. Pour cela, nous proposons la création d'une base de données externe dont nous détaillons ici le contenu. Par *externe*, nous entendons le fait que cette base de données est déconnectée de l'entrepôt de données. De plus, nous adoptons ce mode de stockage relationnel pour plusieurs raisons pratiques. Tout d'abord, ce mode de stockage est bien adapté pour stocker un volume important de connaissances. Ensuite, l'interfaçage entre cette base et un module de mise à jour de cette connaissance est aisé.

Dans la mesure où plusieurs contextes peuvent cohabiter au sein d'un même entrepôt (*e.g.*, la catégorisation d'une tension artérielle et celle d'une posologie), il est nécessaire de stocker quels sont les attributs qui interviennent dans chaque contexte (*i.e.*, quels sont les attributs contextualisants) car ceux-ci peuvent différer pour chaque contexte. Nous proposons alors la création d'une table MTC (Méta-Table des Connaissances) afin de stocker la structure associée à chaque contexte présent dans l'entrepôt.

Définition 6.19 (MTC)

Soit $MTC = (\text{Contexte}, \text{Elément}, \text{Table}, \text{Type})$ une table relationnelle où :

- **Contexte** désigne l'identifiant du contexte IdC_i ;
- **Élément** désigne un attribut intervenant dans le contexte IdC_i ;
- **Table** correspond à la table relationnelle de l'entrepôt où *Element* est instancié ;
- **Type** définit le rôle joué par *Element* dans le contexte c_i . *Type* vaut "Contexte" si *Elément* est un élément contextualisant de IdC_i et vaut "Résultat" si *Elément* est l'élément contextualisé de IdC_i .

Exemple 6.12 *Considérons le cas d'étude présenté dans la section 2 et analysons comment est stockée la structure du contexte associé à la généralisation d'une tension artérielle. Nous rappelons que la catégorisation d'une tension est fonction de plusieurs paramètres : la tension mesurée, la catégorie d'âge du patient et du fait qu'il soit fumeur ou non. Par convention, le nom donné à chaque contexte est celui de la mesure à généraliser. Ici, l'attribut Contexte vaut donc "Tension". Ensuite, les attributs dont la modalité est Contexte (i.e, les attributs contextualisant) sont SubCatAge, Fumeur et Tension. Ceux-ci sont définis dans les tables PATIENTS et TENSIONS. Enfin, l'attribut contextualisé est CatTension. Le tableau de la figure 6.10 présente un extrait de la table MTC associé au contexte Tension.*

MTC			
Contexte	Attribut	Table	Type
Tension	SubCatAge	Patients	Contexte
Tension	Fumeur	Patients	Contexte
Tension	Tension	Tension	Contexte
Tension	CatTension	TC	Résultat
Posologie

FIGURE 6.10 – Extraits de la table MTC décrivant le contexte associé à la généralisation d'une tension artérielle.

Une fois la structure de chaque contexte définie dans MTC, nous nous intéressons à son instanciation. Pour cela, nous définissons au sein de la base de données externe une nouvelle table relationnelle TC (Table des Connaissances). Notons que sur l'exemple précédent, la table associée à l'attribut contextualisé (*CatTension*) est cette nouvelle table TC. Dans la mesure où les valeurs prises par cet attribut sont dépendantes des valeurs prises par les attributs *contextualisants*, ce choix de modélisation semble le plus adapté.

Définition 6.20 (TC) *Soit $TC = (Contexte, Instance_Contexte, Elément, Valeur)$ une table relationnelle de la base de données externe telle que :*

- **Contexte** désigne l'identifiant du contexte (IdC_i);
- **Instance_ Contexte** identifie l'instance du contexte concernée ($Id_{inst_{ij}}$);
- **Attribut** désigne un élément intervenant dans le contexte IdC_i ;
- **Valeur** correspond à la valeur affectée à Attribut dans $Id_{inst_{ij}}$.

Pour diminuer la taille de la table TC, le nombre d'instances de contexte stockées peut être réduit en autorisant l'attribut Valeur à contenir une expression SQL syntaxiquement correcte. Par exemple, “*IN (Oui,Non)*” évite la création de deux instances de contexte (une pour chaque valeur de l'opérateur *IN*).

Considérons le contexte *Tension* présenté lors de l'exemple précédent. Comme illustré dans le tableau 1, de nombreuses règles expertes existent pour déterminer la généralisation d'une tension artérielle. Alors que la structure de ces règles (*i.e.*, quels sont les attributs qui impactent sur la généralisation d'une mesure) est définie au niveau de la méta-table des connaissances MTC, les instances de ces contextes (*i.e.*, les règles expertes) sont définies au niveau de la table des connaissances TC. Le tableau 6.11 présente un extrait du contenu stocké dans TC. Chaque ligne du tableau 1 représente une instance du contexte *Tension*.

Exemple 6.13 *Illustrons le stockage des instances de contexte en considérant la connaissance “chez un nourrisson, une tension supérieure à 12 est élevée”. Cette instance est représentée dans TC par les 4 n-uplets dont la modalité de l'attribut Contexte est Tension et celle de l'attribut Instance_contexte est 1. Parmi ces 4 n-uplets, les 3 premiers permettent de décrire les conditions à réunir pour que la généralisation (stockée dans le quatrième n-uplet) soit valide.*

6.2 Gérer les connaissances expertes

L'approche que nous proposons permet un gain d'expressivité important dans les entrepôts de données médicales en permettant la modélisation, le stockage et l'exploitation de connaissances expertes dans le processus de généralisation de mesures. Néanmoins, un autre aspect essentiel doit être considéré ici : gérer, maintenir et mettre à jour cette connaissance. En effet, en fonction des patients accueillis dans le service ou des progrès de la médecine, certains contextes et instances de contexte peuvent être ajoutés, modifiés ou bien supprimés. En outre, la simple consultation de cette connaissance via une interface simple représente un besoin pour des utilisateurs non-spécialistes en bases de données.

TC			
Contexte	Instance_contexte	Attribut	Valeur
Tension	1	SubCatAge	= Nourisson
Tension	1	Fumeur	IN (Oui,Non)
Tension	1	Tension	>12
Tension	1	CatTension	= Élevée
Tension	2	SubCatAge	= Adulte
Tension	2	Fumeur	= Oui
Tension	2	Tension	>14
Tension	2	CatTension	= Élevée
Tension	3	SubCatAge	= 3ème âge
Tension	3	Fumeur	IN (Oui,Non)
Tension	3	Tension	>16
Tension	3	CatTension	= Élevée
Tension	4

FIGURE 6.11 – Extraits de la table TC présentant quelques instances de contexte associées au contexte Tension.

C'est pour répondre à ces objectifs que nous avons développé une application web basée sur le SGBD Postgresql. Comme nous le verrons dans le chapitre suivant, le choix de ce SGBD est lié aux technologies imposées pour le développement d'une plate-forme de démonstration des principales approches proposées dans le projet MIDAS. Nous détaillons maintenant quelques unes des principales fonctionnalités en nous appuyant sur le cas d'étude présenté dans la section 2.

Puisque la création de la base de données experte ne peut être réalisée sans l'aide des médecins, il est essentiel que l'interface de saisie des contextes et de leurs instances soit la plus simple et intuitive possible. C'est pourquoi l'application web développée propose les fonctionnalités suivantes :

- La création, suppression et modification de contextes,
- La création, suppression et modification d'instances de contexte.

Nous nous attardons sur l'une d'entre elles pour montrer la facilité d'utilisation de notre application. La figure 6.12 présente et développe la partie de l'application gérant les instances de contexte. La partie supérieure affiche les différentes instances pour un contexte donné. Un simple clic sur [*Update*] transforme cette ligne d'affichage en un formulaire permettant la modification de l'instance. Sur le même principe, cliquer sur [*X*] permet de supprimer une instance. La partie inférieure de cette page permet la saisie de nouvelles instances de contexte. Lors de la soumission de ces formulaires, une vérification est effectuée pour s'assurer que deux instances de contexte identiques n'aient pas une généralisation différente.

The screenshot shows a web interface for managing contexts. At the top, there is a table with columns for values, ranges, and generalizations. Below the table is a section titled "DEFINE NEW INSTANCES OF THE CONTEXT blood_pressure" which contains a form with input fields and dropdown menus for creating new instances and generalizations. Annotations (1) through (4) point to specific elements in the interface, and explanatory text on the right describes their functions.

high	> (11)	IN (child,baby,teen)	IN (yes,no)	[Update]	[X]
normal (1)	Between 10 14	Select values senior citizen baby adult	Select values no yes	[Update] (2)	[X] (3)
normal	between (8,16)	IN (adult)	IN (no)	[Update]	[X]
normal	between (11,14)	IN (senior_citizen)	IN (no,yes)	[Update]	[Cancel delete]
normal	between (9,11)	IN (teen,baby)	IN (no,yes)	[Update]	[X]

Mise à jour ou suppression d'instances

- (1) Chaque ligne correspond à une instance
- (2) Cliquer sur [Update] permet de modifier une instance
- (3) Cliquer sur [X] permet de supprimer une instance

Nouvelles instances

- (1) Saisir une nouvelle valeur pour la généralisation
- (2) Cliquer sur [+] permet de créer une nouvelle instance possédant la même généralisation
- (3) Renseigner le formulaire
- (4) Cliquer ici permet de créer un ensemble d'instances possédant la même généralisation

FIGURE 6.12 – Capture d'écran commentée de la page permettant la gestion des contextes.

En conclusion, cette interface web permet une gestion simple, intuitive et efficace de la connaissance.

7 Discussion

Dans ce chapitre, nous nous sommes intéressés à la problématique peu abordée de la généralisation contextuelle d'éléments numériques et avons introduit pour cela un nouveau type de hiérarchies, les hiérarchies contextuelles. Afin d'intégrer cette nouvelle catégorie de hiérarchie dans la modélisation d'un entrepôt de données, nous avons proposé un modèle conceptuel multidimensionnel pour définir un entrepôt comportant des hiérarchies contextuelles. Afin d'assurer une interaction optimale entre le concepteur et le futur utilisateur de l'entrepôt, un modèle graphique est proposé pour représenter clairement les différents composants de l'entrepôt. Parmi les particularités supplémentaires associées à ce modèle, nous pouvons citer la possibilité de définir des hiérarchies sur des mesures associées à des faits. Nous proposons ensuite les outils nécessaires pour la manipulation des hiérarchies contextuelles en détaillant les processus inhérents à la généralisation et la spécialisation contextuelles. Enfin, nous adoptons une modélisation logique de type ROLAP pour stocker les différents contextes et instances présents dans un entrepôt de données contextuels. En effet, en s'appuyant sur une base de connaissance externe, il est possible de matérialiser de façon générique ces connaissances et de les exploiter efficacement. Nous proposons en outre un outil pour mettre à jour ces connaissances expertes. Pour résumer, nous proposons, dans ce chapitre, un nouveau modèle d'entrepôts de données permettant un gain important d'expressivité ainsi qu'une plus grande flexibilité dans la gestion des connaissances stockées ainsi que les

premiers outils pour exploiter un tel entrepôt. Malgré ces avantages indéniables, de nombreuses perspectives restent associées à ce travail pour aller toujours dans le sens d'une plus grande adéquation entre la modélisation proposée et la situation réelle.

Tout d'abord, une première série de perspectives concerne la modélisation conceptuelle de l'entrepôt contextuel. Pour l'instant, nous supposons que l'ensemble des éléments contextualisants est essentiellement composé d'attributs de dimensions (excepté quand il s'agit d'une hiérarchie contextuelle de mesures auquel cas il existe une mesure contextualisante). Bien que cette solution permette de modéliser un grand nombre de situations (*e.g.* la dépendance entre la normalité d'une tension et l'âge du patient), il existe des situations qui ne peuvent être modélisées. Par exemple, il n'est pas possible de modéliser le fait que la catégorisation d'une tension dépend également des médicaments prescrits auparavant. Conceptuellement, les modifications à apporter au modèle pour représenter ce type de contexte sont très légères. Néanmoins, la prise en compte de tels contextes a des implications sur les techniques proposées pour manipuler ce type de hiérarchies. Pour illustrer les modifications qu'il faudrait apporter à notre proposition, considérons que nous souhaitons modéliser la connaissance suivante : *une tension artérielle supérieure à 14 est très élevée si le patient s'est vu prescrire un médicament hypotenseur deux heures avant la mesure de sa tension artérielle*. Alors que pour une tension et un patient donnés correspond toujours la même généralisation dans le modèle que nous proposons, cela ne sera plus le cas avec une telle connaissance. Intuitivement, il faudrait donc rechercher dans une autre table de faits (*i.e.* la table *Posologie* en l'occurrence), si le patient dont on veut généraliser la tension s'est vu prescrire un médicament hypotenseur au plus tard deux heures avant la prise de tension que l'on cherche à généraliser. Outre ces conséquences algorithmiques, il faudrait également s'interroger sur les conséquences en terme de stockage de telles connaissances.

Par ailleurs, nous sommes convaincus que l'intégration de la logique floue [ZAD88] dans notre proposition entraînerait un gain de flexibilité important dans la gestion de la connaissance experte. Selon nous, la logique floue peut être introduite de trois manières distinctes que nous détaillons brièvement ici :

- En autorisant que des hiérarchies classiques soient des hiérarchies floues (*e.g.* un patient appartient un peu à la catégorie *Jeune Enfant* et beaucoup à la catégorie *Enfant*) ;
- En autorisant l'existence de critères flous dans les instances de contexte (*e.g.* si un patient plutôt âgé et fumant beaucoup a une tension supérieure à 14 alors celle-ci est considérée comme élevée) ;
- En associant une mesure de confiance pour chaque instance de contexte (*e.g.* l'instance 1 du contexte 1 est certaine à 90%).

Dans toutes les situations précédemment décrites, une généralisation contextuelle ne serait alors plus unique mais nous disposerions d'un ensemble de généralisations possibles avec leur *score*

associé (*e.g.* pour un patient et une tension donnés, le score associé à *Elevé* est 0,8 et le score associé à *Normal* est 0,4). Plusieurs questions sous-jacentes apparaissent alors. Comment déterminer ces scores? Comment combiner les différentes situations décrites? Enfin, quelles sont les conséquences sur notre proposition (*i.e.* sur la modélisation, les techniques de manipulation des hiérarchies contextuelles et sur la mise en œuvre de tels entrepôts contextuels flous)?

Une deuxième série de perspectives associées à ce travail concerne l'interrogation de tels entrepôts. Clairement, si l'on se restreint à la situation où il ne peut exister de hiérarchies de mesures (mais qu'une hiérarchie d'attributs de dimension peut être contextuelle) et dans la mesure où nous fournissons les outils pour généraliser et spécialiser contextuellement, les possibilités de requêtes offertes par notre modèle sont identiques aux entrepôts "*classiques*". Par contre, l'introduction de hiérarchies sur des mesures conduit à s'interroger sur l'exploitation de ces mesures généralisées. En effet, ces mesures généralisées étant, par définition, textuelles, elles ne possèdent pas les propriétés d'additivité ou de semi-additivité permettant de les exploiter telles des mesures numériques traditionnelles. Dès lors, s'intéresser à des travaux portant sur l'agrégation de données textuelles [RTT07] constitue une perspective intéressante.

Enfin, la troisième série de perspectives associée à ce travail concerne la mise en œuvre physique des entrepôts de données contextuels et particulièrement de la base de connaissances expertes. Tout d'abord, une telle base de règle doit posséder un moteur de vérification pour assurer la consistance des règles stockées [SSS82, PSB92]. En effet, nous avons pour l'instant supposé que l'ensemble des règles était consistant mais en pratique, un utilisateur² peut saisir des règles non cohérentes entre elles. Même si, dans le prototype développé, nous interdisons que deux règles possédant les mêmes prémices aient une conclusion différente, nous sommes convaincus de la nécessité d'utiliser un tel système de vérification. De plus, même si la complétude théorique semble difficile à garantir, il faudrait s'assurer, *a minima*, de la complétude de cet ensemble de règles par rapport aux instances des dimensions et faits de l'entrepôt.

Une autre perspective associée à la matérialisation de cette connaissance experte serait d'étudier la possibilité de représenter les instances associées à un contexte sous la forme d'un arbre de décision [Bre84, Qui93]. Intuitivement, les concepts manipulés présentent des similarités. Une instance d'un contexte peut en effet être perçue comme un chemin dans un arbre de décision conduisant à la généralisation correcte. Dès lors, il apparaît pertinent de confronter la matérialisation adoptée dans ce chapitre (via une base de données relationnelle externe) à une matérialisation sous forme d'arbres. Cela impliquerait alors de se poser les questions suivantes : (1) quel est l'im-

2. Ce risque s'accroît considérablement dans un environnement multi utilisateurs

l'impact sur la généralisation et d'une spécialisation contextuelles ?

Enfin, d'un point de vue pratique, il serait bénéfique d'automatiser en partie la définition des contextes et des instances associées grâce à des approches de fouilles de données exploitant des bases de connaissances ou internet. Le protocole adopté serait alors étroitement lié au domaine d'application concerné par l'entrepôt de données.

Chapitre 7

Implantation dans une plate-forme de démonstration

1 Introduction

Ce travail de thèse s'est déroulé dans le cadre d'un projet ANR dénommé MIDAS (ANR-07-MDCO-008). Un des objectifs du projet était la création d'une plate-forme de démonstration intégrant les principales approches proposées dans le cadre de ce projet. Cette plate-forme permet ainsi de confronter différentes techniques de résumé mises au point par les partenaires académiques à des jeux de données fournis par les partenaires industriels du projet. L'objectif de ce chapitre est de présenter l'architecture générale de cette plate-forme.

Cette plate-forme s'appuie et étend les fonctionnalités du DSMS *Esper*. Nous rappelons en effet qu'un DSMS ne permet pas, par définition, de construire un résumé de flot de données mais est particulièrement adapté au traitement des requêtes continues. Nous présentons ainsi comment ce DSMS a été étendu pour permettre d'accueillir des approches de résumés de flots de données. Nous verrons que la mise au point de cet outil de démonstration a nécessité une certaine genericité concernant les entrées des algorithmes implantés. Cette genericité en entrée a été permise grâce à l'obligation pour les approches de fournir des fichiers XML (eXtensible Markup Language) à la structure communément définie. En sortie, le stockage des résumés produits est assuré par le SGBD Postgresql. De part les grandes différences qui existent entre les différentes approches, la structure relationnelle des bases hébergeant ces résumés ne peut être générique.

Outre l'extension du DSMS *Esper* pour permettre la construction de résumé de flot de données, nous souhaitons également fournir un outil de visualisation en temps réel du flot de données. Aussi, nous proposons une application web basée sur les technologies *Flex* et *BlazeDS* pour réa-

liser cette visualisation. Le fonctionnement de cette interface est décrit dans la suite de ce chapitre.

Ce chapitre est donc organisé de la façon suivante. Dans la section 2, nous présentons l'architecture générale de la plateforme et détaillons les mécanismes permettant d'assurer la généricité en entrée des algorithmes implantés. La section 3 est consacrée à la présentation de l'outil de visualisation associé à cette plate-forme. Nous indiquons ensuite quelles approches proposées dans ce travail ont été implantés et en quoi elles permettent de répondre aux besoins applicatifs spécifiés par les partenaires industriels. Une conclusion résumant les caractéristiques de cette plate-forme est dressée dans la dernière section.

2 Architecture de la plateforme MIDAS

La présente section vise à introduire l'architecture de la plateforme MIDAS. Pour commencer, l'architecture de la plateforme est décrite dans la section 2.1. La configuration de la plateforme est introduite dans la section 2.2. Cette configuration est effectuée à travers des fichiers XML spécifiant les divers paramètres à la plateforme. Enfin, la gestion de la base de données des résumés est décrite dans la section 2.3.

2.1 Vue d'ensemble de l'architecture de la plateforme MIDAS

La figure 7.1 décrit une vue globale de l'architecture de la plateforme MIDAS. Les flot de données provenant de différentes sources (fichiers CSV, sockets, etc.) sont interceptés par le DSMS Esper qui se charge de l'exécution des requêtes continues sur le contenu de ces flot. Ce requêtage est effectué grâce au langage *EPL* (*Event Processing Language*) proposé par Esper.

Les algorithmes de résumé de flot de données interagissent avec Esper afin de récupérer les résultats des requêtes continues en vue de les traiter. Ces algorithmes peuvent se servir d'une base de données pour le stockage des résumés qu'ils génèrent ainsi que pour récupérer des données statiques additionnelles dont ils ont, par exemple, besoin pour compléter les données du flot. Ces algorithmes peuvent, en outre, envoyer des données vers un outil de visualisation (*Midas Stream Visualizer*) afin d'avoir une représentation graphique des données en temps réel. La configuration de la plateforme couvre les points suivants :

- Déclaration de la source et du schéma du flot d'entrée ;
- Instantiation et démarrage du serveur Esper ;
- Initialisation de la connexion à la base de données des résumés ;
- etc.

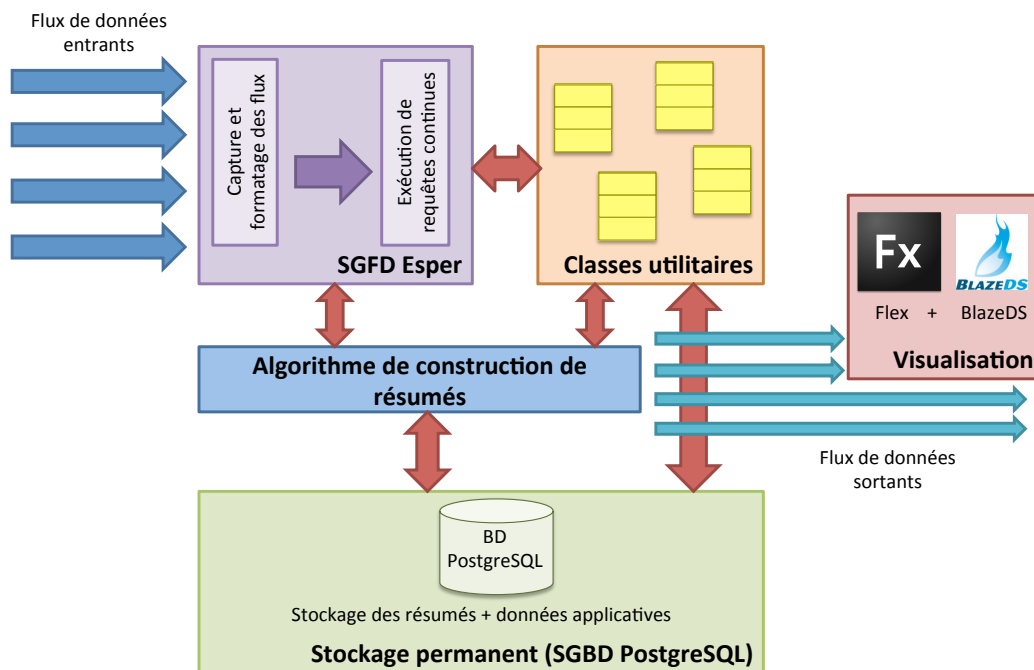


FIGURE 7.1 – Vue d'ensemble de l'architecture de la plateforme MIDAS.

Ces aspects sont gérés grâce à deux fichiers XML qui sont fournis en entrée à la plateforme et qui serviront à spécifier les différents paramètres. Ces fichiers XML sont décrits en détail dans la section 2.2.

2.2 Fichiers XML d'entrée

Le premier fichier XML sert à décrire la structure du flot utilisée par l'algorithme qui sera exécuté sur la plateforme. Le deuxième fichier sert à passer les divers paramètres propres à l'exécution de l'algorithme (*e.g.* noms des champs à connotation spécifique, paramètres de la connexion à la base de données).

Fichier de spécification de la structure du flot

La structure du flot qui va être réceptionné par le serveur Esper doit être indiquée au préalable dans un fichier XML bien structuré. La structure de ce fichier doit obéir au schéma XML illustré dans la figure 7.2.

Les éléments à inscrire dans le fichier XML sont les suivants :

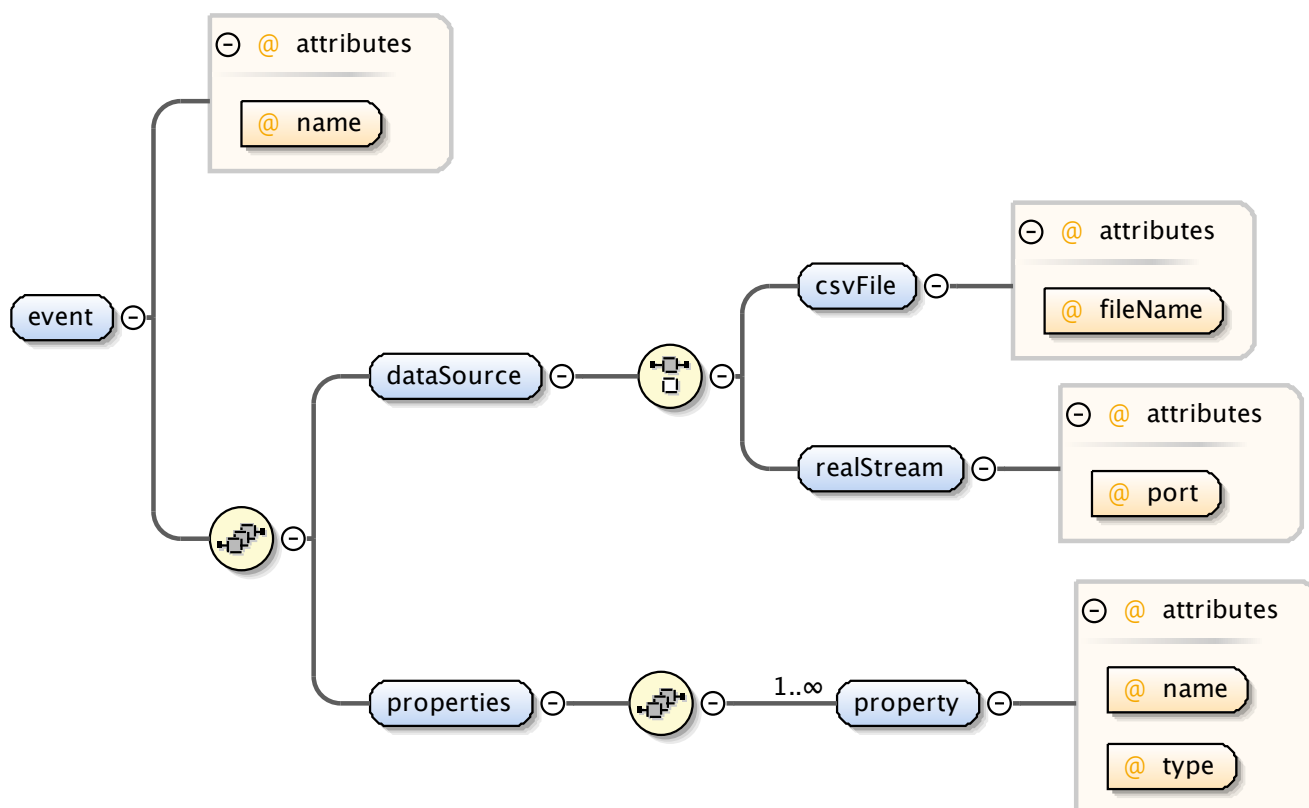


FIGURE 7.2 – Schéma XML pour la spécification de la structure du flot en entrée du serveur Esper

- L'élément `event` est la racine de l'arborescence du fichier XML. L'attribut `name` de cet élément sert à indiquer le nom du flot qui va être reçu par la plateforme. Le nom du flot sera ensuite utilisé pour nommer les tables de résumés dans la base de données (voir section 2.3) ;
- L'élément `dataSource` sert à indiquer la source du flot, deux choix sont possibles :
 1. Un fichier CSV (élément `csvFile`) : dans ce cas, il est obligatoire d'indiquer le chemin d'accès et le nom du fichier (attribut `fileName`) ;
 2. Données sur socket (élément `realStream`) : le port sur lequel Esper va se mettre en écoute doit obligatoirement être renseigné (attribut `port`) ;
- Les champs du flot sont indiqués à travers des éléments `property`, et sont tous regroupés sous l'élément `properties`. Pour chaque champ, il est obligatoire d'indiquer :
 - Le nom (attribut `name`) ;
 - Le type (attribut `type`), qui doit obligatoirement être l'un des types suivants :
 - `int` : entier ;
 - `short` : entier court ;
 - `long` : entier long ;
 - `float` : nombre en virgule flottante ;
 - `double` : nombre en virgule flottante avec double précision ;

- `string` : chaîne de caractères ;
- `boolean` : booléen ;

Un exemple de fichier XML respectant le schéma est présenté dans la figure 7.3. Il décrit les données d'appels téléphoniques. Pour chaque appel, un identificateur et une estampille temporelle sont fournis avec, en plus, des informations sur le client (adresse, nom, sexe, etc.).

```
<?xml version="1.0" encoding="UTF-8"?>
<event name="calls">
  <dataSource>
    <csvFile fileName="./input/exemple.csv"/>
  </dataSource>
  <properties>
    <property name="id" type="int"/>
    <property name="timestamp" type="long"/>
    <property name="usage_type" type="long"/>
    <property name="geographic_zone" type="String"/>
    <property name="call_type" type="String"/>
    <property name="client_postal_code" type="String"/>
    <property name="client_department" type="String"/>
    <property name="client_status" type="String"/>
    <property name="client_name" type="String"/>
    <property name="client_title" type="String"/>
    <property name="client_sex" type="String"/>
    <property name="client_date_birth" type="String"/>
    <property name="client_date_creation" type="String"/>
    <property name="client_age" type="String"/>
    <property name="client_anciennete" type="String"/>
    <property name="date" type="String"/>
    <property name="call_duration" type="int"/>
  </properties>
</event>
```

FIGURE 7.3 – Exemple d'un fichier XML de description de structure de flot lu à partir d'un fichier CSV

Fichier de paramétrage d'exécution

Divers aspects doivent être configurés avant que la plateforme puisse être utilisée. Ces aspects couvrent (1) les paramètres de l'algorithme de résumé de flot qui va être exécuté, (2) l'indication des champs spéciaux présents dans le flot (*i.e.* champs ayant une signification spéciale pour l'algorithme) et (3) les paramètres de connexion à la base de données pour le stockage des résumés. Tous ces paramètres doivent être renseignés dans un fichier XML de configuration dont la structure doit être conforme au schéma illustré dans la figure 7.4 :

- L'élément `configuration` joue le rôle de la racine du fichier XML, il renseigne sur :

1. Le nom de l'algorithme qui va être exécuté sur la plateforme (attribut `algorithmName`).
Le nom de l'algorithme sera aussi utilisé pour nommer les tables de résumés dans la base de données (cf. section 2.3);
 2. Le nom du fichier contenant la structure du flot (attribut `streamSchemaFile`);
 3. Le nom du fichier contenant la configuration de base du serveur Esper (attribut `esperConfiguratio`);
- L'élément `parameters` regroupe les divers paramètres dont l'algorithme a besoin pour s'exécuter. Chaque paramètre est spécifié dans un élément de type `parameter` où il faut indiquer :
 1. Son nom (attribut `name`);
 2. Son type (attribut `type`, les types acceptés étant les mêmes que ceux décrits dans la section 2.2);
 3. Sa valeur (attribut `value`).
 - Si le flot contient un champ spécial qui va jouer le rôle de timestamp, alors ce dernier est indiqué par la présence d'un élément de type `timestamp`¹ où le nom du champ est renseigné;
 - L'élément `specialFields` regroupe les noms des champs ayant une signification particulière pour l'algorithme. Pour chaque champ spécial (élément `field`) sont indiqués les noms avec lesquels il est référencé dans l'algorithme (attribut `nameInAlgorithm`) et dans le flot (attribut `nameInStream`). Ceci permet de découpler la façon dont les champs sont référencés du côté de la source du flot et du côté de l'algorithme qui va réceptionner ce flot.

Un exemple d'un fichier XML validé par ce schéma est présenté dans la figure 7.5. Ce fichier indique qu'il y a un seul paramètre, nommé `threshold` que l'algorithme va utiliser. Il indique également la présence d'un champ spécial nommé `call_duration` que l'algorithme interprète comme un champ pour le filtrage.

Remarque Si l'algorithme nécessite une description de structure XML décrite dans un fichier secondaire (ex. cas des données hiérarchiques), le nom de ce fichier pourra être mentionné dans l'élément paramètres sous la forme d'une chaîne de caractères. Par exemple :

```
<parameter name = "fichierStructure" type = "String" value = "./fichier.xml" />
```

1. L'élément "timestamp" ne sera utilisé par Esper que dans le cas des fichiers CSV et pour les requêtes sur fenêtres seulement. Si l'élément est renseigné, il faut qu'un élément "property" avec le même nom existe dans le fichier XML de spécification de la structure du flot.

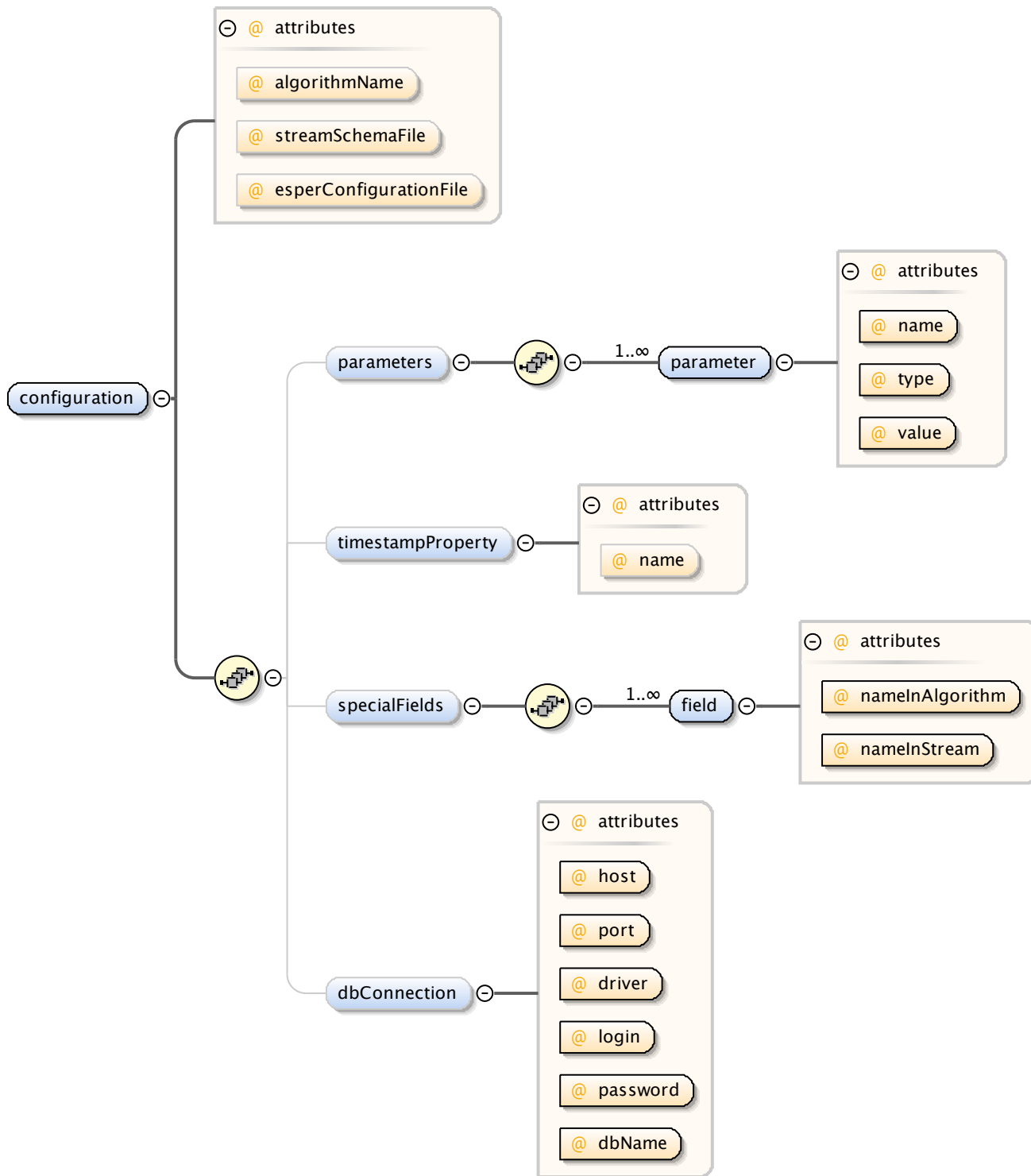


FIGURE 7.4 – Schéma XML pour le paramétrage de la plateforme MIDAS

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration algorithmName="demo4"
  streamSchemaFile="./input/structureCSV.xml"
  esperConfigurationFile="esper.cfg.xml">

  <parameters>
    <parameter name="threshold" type="int" value="10"/>
  </parameters>

  <specialFields>
    <field nameInStream="call_duration"
nameInAlgorithm="filterProperty"/>
  </specialFields>

  <dbConnection host="jules.enst.fr" port="5432"
driver="jdbc:postgresql"
  login="postgres" password="demo" dbName = "dbstream"/>

</configuration>
```

FIGURE 7.5 – Fichier XML pour le paramétrage de la plateforme MIDAS (avec connexion à une base de données)

2.3 Gestion des tables de la base de données pour le stockage des résumés

Les algorithmes de résumés de flux de données sont amenés à stocker leurs résultats (*i.e.* les résumés) dans une base de données. Deux tables standard sont déjà créées et mises à disposition des utilisateurs de la plateforme afin de stocker les informations relatives à l'exécution des algorithmes :

- La table **streams** permet de tracer les informations sur la structure des flux de données sur lesquels les résumés ont été construits ;
- La table **infos_algos** sert à conserver les informations sur l'exécution des algorithmes telles que les dates de début et de fin d'exécution, le nom de l'algorithme et celui du flux, etc.

Un exemple des données stockées dans ces deux tables est montré dans les figures 7.6 et 7.7. La table **streams** indique que deux types de flux ont été traités par la plateforme (l'un contenait des données de trajectoires tandis que l'autre contenait des données d'appels téléphoniques). La table **infos_algos** montre que cinq instances d'un algorithme nommé **demo4** ont été lancées et renseigne sur les paramètres, les dates de lancement et de fin et les flux sur lesquels l'algorithme a effectué ses traitements.

	id integer	name text	schema text
1	1	calls	client_sex TEXT, client_postal_code TEXT, client_status TEXT, geographic_zone TEXT, client_department TEXT, client_date_birth TEXT,
2	2	trajectories	id BIGINT, nextNodeX DOUBLE PRECISION, nextNodeY DOUBLE PRECISION, speed DOUBLE PRECISION, action TEXT, repNum BIGINT,

FIGURE 7.6 – Exemple du contenu de la table "streams"

	num integer	algo text	start text	stop text	param text	streamid integer
1	1	demo4	22 nov. 2010 16:32:407	22 nov. 2010 16:32:01	threshold: 30, filterProperty: call_duration	1
2	2	demo4	22 nov. 2010 16:33:725	22 nov. 2010 16:33:367	threshold: 30, filterProperty: call_duration	1
3	3	demo4	22 nov. 2010 16:33:842	22 nov. 2010 16:33:469	threshold: 10, filterProperty: call_duration	1
4	4	demo4	22 nov. 2010 16:33:79	22 nov. 2010 16:33:672	threshold: 100.0, filterProperty: speed	2
5	5	demo4	22 nov. 2010 16:35:670	22 nov. 2010 16:35:278	threshold: 150.0, filterProperty: speed	2
6	6	demo4	25 nov. 2010 10:57:778	25 nov. 2010 10:57:389	threshold: 10, filterProperty: call_duration	1
7	7	demo4	25 nov. 2010 10:58:48	25 nov. 2010 10:58:666	threshold: 10, filterProperty: call_duration	1

FIGURE 7.7 – Exemple du contenu de la table "infos_algos"

D'autres tables sont nécessaires pour le stockage des résumés générés par les algorithmes. La structure de ces tables varie en fonction de l'algorithme utilisé et de la nature du flux qu'il reçoit. C'est donc à chaque algorithme qu'incombe la tâche de gérer la création, l'insertion et les mises à jour à effectuer sur ses propres tables. Cependant, les noms attribués à ces tables doivent respecter une certaine convention de nommage : le nom de la table doit être sous la forme *<nom de l'algorithme>_<nom du flux>_<suffixe>_<numéro de séquence>*. Le suffixe décrit le rôle de la table (ex. dans StreamSamp, le suffixe `events` désigne la table où les données échantillonnées sont stockées tandis que le suffixe `samples` indique la table qui stocke les métadonnées sur les échantillons), si le suffixe n'est pas spécifié par l'algorithme, le suffixe `events` sera utilisé par défaut. Le numéro de séquence référence l'identificateur de l'instance d'exécution de l'algorithme dans la table `infos_algos`.

3 Description de l'outil de visualisation

Plutôt qu'une description poussée sur les technologies utilisées, nous abordons la présentation de cet outil par la description de ses fonctionnalités. Cette partie présente les différentes interfaces de l'application. Nous allons avoir un aperçu sur la navigation entre les menus et la manipulation de l'interface pour aboutir à une visualisation du flot de données et de leurs résumés.

3.1 Menu principal

En se connectant à l’adresse de “MIDAS Stream Visualizer”, nous accédons à la page principale de l’application. Ce menu résume les étapes que l’utilisateur doit suivre pour visualiser un flot de données comme suit :

Choix du flot à afficher (“Data Stream”);

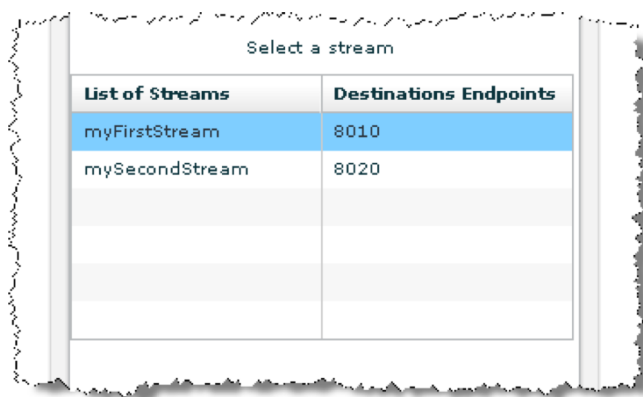
Choix du type de graphique à utiliser (“Chart”);

Lancement de la visualisation du flot de données (“Show Stream”);

Chaque sous-menu est détaillé dans les paragraphes suivants.

3.2 Choix du flot à afficher

L’application offre à l’utilisateur le choix de sélectionner un flot parmi les flots que le DSMS Esper envoie en sortie. Ces flots se présentent sous forme d’une liste contenant le nom du flot de données et la destination vers laquelle le flot est envoyé à partir de Esper. Pour choisir un flot, il suffit de sélectionner un élément de cette liste (*c.f.* figure 7.8).



List of Streams	Destinations Endpoints
myFirstStream	8010
mySecondStream	8020

FIGURE 7.8 – Sélection d’un flux de données

3.3 Sélection du type de graphique à utiliser

Trois types de graphique sont proposés à l’utilisateur (*c.f.* figure 7.9) :

- “Time Curve” : Dans ce mode d’affichage, la (ou les) série(s) de données se présente(nt) sous forme de courbe(s) ayant comme axe horizontal le timestamp des données. Il s’agit donc d’une courbe linéaire d’un attribut choisi en fonction du temps. Par exemple, l’utilisation de ce type de graphique permet d’afficher l’évolution des valeurs d’un capteur au cours du temps ;

- “Non Linear Curve” : Ce mode d’affichage présente l’évolution des données au cours du temps selon les attributs associés aux axes horizontal et vertical. A la différence du premier mode, les données sont non linéaires. Par exemple, l’utilisation de ce type de graphique permet d’afficher la trajectoire d’un mobile au cours du temps dans un espace de coordonnées (x,y) ;
- “Column Chart” : Ce mode présente des données sous forme de colonnes verticales (chaque colonne présente une valeur d’un attribut choisi du flot) de hauteur variable au cours du temps.

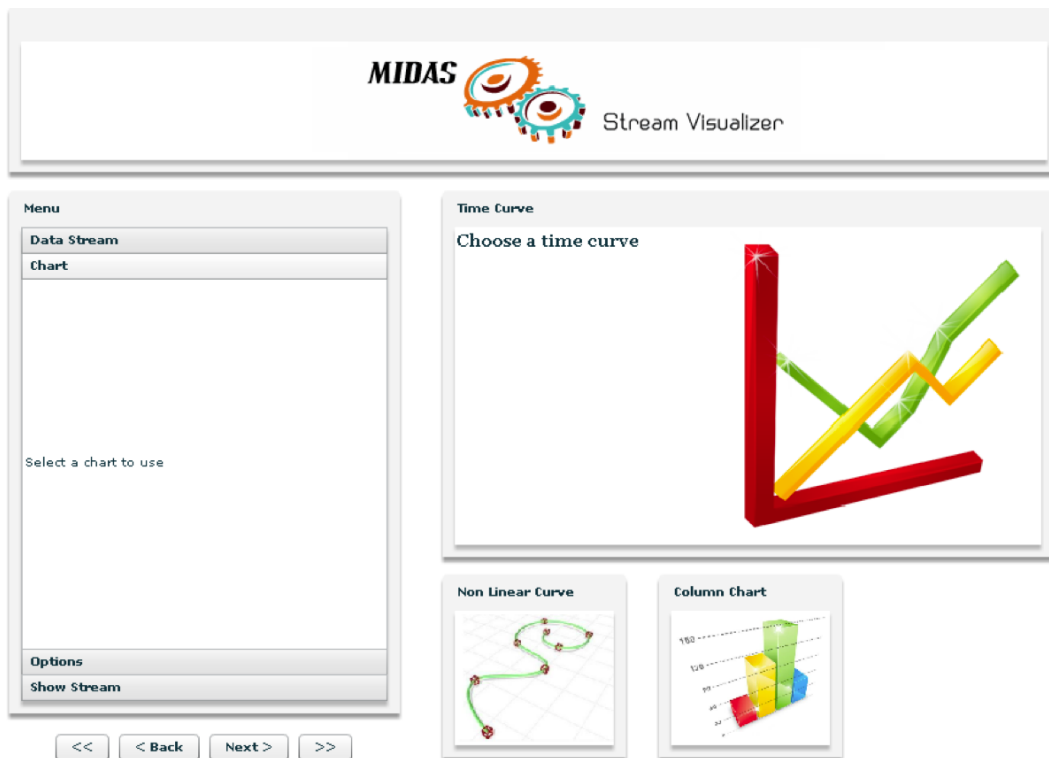


FIGURE 7.9 – Sélection du type de graphique

L'utilisateur peut appliquer des filtres sur le flot de données afin de visualiser des séries multiples. Ainsi, il choisit le nom d'attribut sur lequel il va appliquer le filtre et définit ensuite une liste de valeurs de sélection.

3.4 Lancement de la visualisation

Après avoir choisi le graphique à utiliser et spécifié les paramètres d’affichage appropriés, l'utilisateur peut lancer la visualisation du flot en passant au menu “Show Stream”. L'utilisateur est amené à cliquer sur le bouton de visualisation situé dans ce menu comme l'indique la figure 7.10. A cette étape, le client Flex se connecte au service d'envoi de messages (contenant les nouveaux

événements) à travers la destination déjà choisie dans le menu “Data Stream”. Ce service tourne en arrière plan et est assuré par le DSMS Esper.

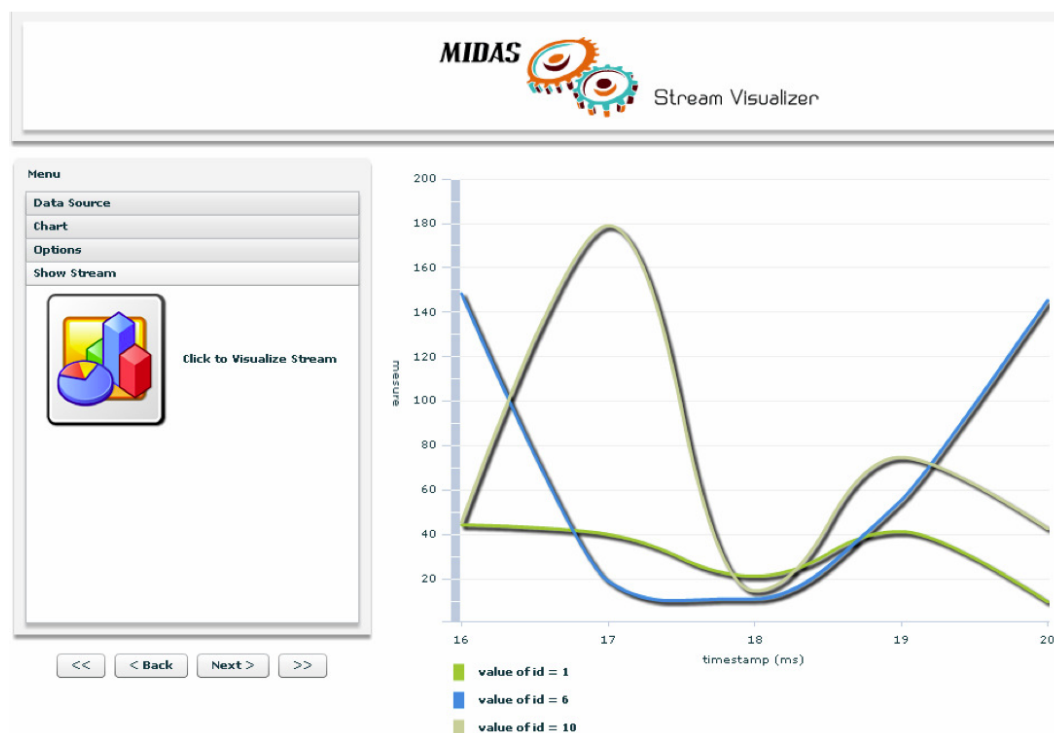


FIGURE 7.10 – Exemple de visualisation fourni par l’outil. Ici, les valeurs prises par trois attributs au cours du temps sont affichées

4 Approches implantées et besoins applicatifs associés

4.1 Analyse d’usages de clients d’Orange

Orange est un acteur majeur dans le secteur de la téléphonie mobile et a su s’adapter à l’avènement des *web phones*. Parmi les services proposés, leur portail web mobile permet la consultation des actualités, l’accès à du contenu vidéo ou à l’espace client, . . . L’analyse des logs de navigation se révèle alors stratégiquement cruciale car une meilleure connaissance des usages des utilisateurs présente de nombreux intérêts aussi pour l’utilisateur que pour l’entreprise. Côté utilisateur, cette meilleure connaissance permet par exemple la mise en place d’un système de recommandation plus adapté. Côté entreprise, déterminer les séquences fréquentes de navigation permet, par exemple, de mieux ajuster les offres publicitaires.

Depuis une dizaine d'années environ, le *web usage mining* et particulièrement la découverte de séquences de navigation fréquentes connaît un intérêt croissant [SCDT00, BM98, MPT00]. Dès lors, il est légitime de se demander ce que peuvent apporter les approches de résumé de flot dans ce contexte. Un élément de réponse provient de la volumétrie des fichiers de logs gérés. Quotidiennement, 10 Go de données sont générés. Ce volume, bien que conséquent, peut être raisonnablement traité par une approche traditionnelle. Cependant, certaines analyses souhaitées nécessitent de considérer un historique beaucoup plus important. Par exemple, si une analyse concerne les trois derniers mois, il faudra considérer $3 \times 30 \times 10Go = 1To$. Bien qu'efficaces, les techniques de fouilles de données issues de données web ne passent pas à l'échelle quand il s'agit de manipuler un tel volume de données.

Pour répondre à cette problématique, nous avons implanté l'approche *ALIZES* dans le démonstrateur. De par la sémantique du résumé produit (*i.e.* un résumé constitué de l'agrégation de séquences fréquentes sur une partie de l'historique), *ALIZES* est particulièrement adapté à l'analyse de l'évolution des habitudes des utilisateurs. En outre, la prise en compte de la multidimensionnalité et des hiérarchies associées aux différentes pages du portail web permet (1) de mieux cibler les comportements fréquents grâce à l'intégration de dimensions décrivant l'abonné (*e.g.* âge, ville de résidence) et (2) de découvrir des séquences multiniveaux décrivant plus fidèlement et plus durablement le comportement fréquent des utilisateurs.

4.2 Résumé de flots de données de capteurs

Dans le cadre de la maintenance des installations de production d'EDF, des études de surveillance et de diagnostic des matériels sont régulièrement effectuées. Dans ce domaine, de nouvelles approches basées sur une analyse globale de données de fonctionnement se développent aujourd'hui (et sont appelées "méthodes de surveillance empiriques"). Ces méthodes permettent d'identifier des modèles de comportement « normal » de machines ou de *process* puis de détecter des "écarts" significatifs à ces modèles et ainsi aider au diagnostic de défaut. Plusieurs méthodes de surveillance empirique existent aujourd'hui, basées sur différents algorithmes, tels que l'analyse en composantes principales (ACP), la régression polynomiale, ou encore l'utilisation de réseaux de neurones. Dans tous les cas, cette surveillance empirique des matériels s'effectue hors ligne et exploite des données historisées caractéristiques de l'état du système et issues de capteurs implantés sur les installations de production.

Une voie d'amélioration importante est la mise en oeuvre d'un algorithme de surveillance empirique qui soit directement implémentable sur un flot de données et permette ainsi la détection d'anomalies de fonctionnement au plus près de leur apparition : l'objectif est d'effectuer un ap-

prentissage permanent avec une mise à jour du modèle d'apprentissage à chaque nouvelle donnée disponible, et de toujours analyser avec le dernier modèle disponible. Cela permettrait à la fois de résoudre des problèmes de volumétrie, et d'améliorer la représentativité des données.

Ainsi pour répondre à ce besoin, nous avons implanté l'approche décrite dans le chapitre 4 pour résumer ce flot de données de capteurs. Comme nous l'avons constaté dans les expérimentations menées dans le chapitre en question, il est possible de restituer très fidèlement les courbes d'origine via cette méthode.

5 Discussion

Dans ce chapitre, nous avons présenté la plate-forme de démonstration développée dans le cadre du projet MIDAS. L'objectif de cette plate-forme est de permettre l'utilisation de différentes techniques de résumé sur le même jeu de données et inversement. Nous avons également présenté un outil connexe à cette plate-forme : l'outil de visualisation des flots et des résumés produits. Enfin, nous avons précisé quelles sont les approches décrites dans ce manuscrit qui ont été implantées dans cette plate-forme et en quoi elles pouvaient répondre aux besoins des partenaires industriels.

Les perspectives associées à cette plate-forme sont de deux ordres. Dans un premier temps, il s'agira, pour les industriels, de tester cet outil et de vérifier empiriquement et en pratique l'adéquation des approches pour répondre à leur besoin. Ensuite, de par la genericité en entrée des méthodes implantées, il peut être envisagé de rendre cet outil public afin d'offrir à la communauté un moyen d'intégrer et de confronter dans une même solution logicielle différentes techniques de résumé.

Chapitre 8

Bilan général et perspectives

*L'esprit qui invente est toujours mécontent
de ses progrès, parce qu'il voit au-delà.*

JEAN LE ROND D'ALEMBERT

Pour conclure ce mémoire, nous proposons de faire un bilan de nos contributions et de discuter des perspectives associées à cette thèse.

Bilan général

Nous avons vu dans la discussion associée à l'état de l'art des approches de construction de résumé de flots de données que les approches existantes exploitaient peu ou pas les hiérarchies associées aux données.

Ce constat a motivé les travaux menés au cours de cette thèse et nous a ainsi conduit à proposer des solutions pour exploiter davantage et efficacement les hiérarchies. Pour aborder cette problématique, nous avons mis en place trois stratégies :

- Construire des résumés en exploitant l'intégralité des données circulant sur le flot ;
- Coupler des approches d'extraction de motifs fréquents à des mécanismes de généralisation pour ne retenir que les événements fréquents ;
- Proposer une nouvelle catégorie de hiérarchies pour accroître la qualité des résumés proposés.

Construction de cubes de données alimentés par un flot de données

Dans le chapitre 2, nous avons vu qu'il n'existe, à notre connaissance, qu'une seule approche (*i.e.* Stream Cube [HCD⁺05]) permettant la construction d'un cube de données alimenté par un flot et prenant en compte les hiérarchies. La discussion associée à cette approche (*c.f.* page 43) a mis en avant trois limites auxquelles nous avons répondu dans les chapitres 3 et 4.

Les deux premières limites associées à Stream Cube concernent le stockage d'informations redondantes pas nécessairement utilisées et un temps de mise à jour potentiellement pénalisant. Aussi, nous proposons, dans le chapitre 3, d'étendre le principe des *titled time windows* à toutes les dimensions hiérarchisées. Dès lors, chaque partie de l'historique du flot n'est conservée qu'au niveau suffisant pour répondre à la majorité des requêtes utilisateur. Nous introduisons pour cela les fonctions de précision qui définissent, pour chaque niveau des dimensions, quelle partie de l'historique il est pertinent de stocker. Cette extension des *titled time windows* permet de définir *Window Cube*, une structure de résumé très compacte et qui se met rapidement à jour puisqu'elle minimise les propagations des données à des niveaux plus généraux. Cependant, la qualité (en terme de représentativité) du résumé produit dépend fortement des fonctions de précision. Nous proposons alors une technique de définition automatique exploitant les logs de requêtes utilisateurs. Enfin, face à l'impossibilité pour un résumé de répondre exactement à toutes les requêtes portant sur l'historique d'un flot, nous proposons de générer des requêtes alternatives pour fournir, malgré tout, une réponse pertinente à l'utilisateur.

L'utilisation des *titled time windows* pour intégrer et compresser la dimension temporelle dans les cellules d'un cube entraîne une dégradation rapide de la qualité du résumé. En particulier, dans la section 5.2 du chapitre 2, nous discutons de l'inadéquation de ce modèle pour résumer l'historique d'éléments apparaissant peu fréquemment ou périodiquement. Aussi, nous proposons, dans le chapitre 4, une structure de liste dynamique pour conserver l'historique des éléments précis du flot et rompre ainsi avec les agrégations systématiques des *titled time windows*. Cette structure présente la particularité de ne déclencher une agrégation qu'en fonction de la proximité temporelle des éléments stockés et permet ainsi de retenir durablement l'historique d'éléments rares. Stocké dans une structure de graphe, nous montrons expérimentalement que ce résumé répond aux trois critères essentiels dans notre contexte.

Utilisation conjointe des motifs fréquents et des hiérarchies

Dans le chapitre 2, nous avons vu que les techniques d'extraction de motifs fréquents peuvent être utilisées à des fins de résumé et avons nuancé cette affirmation par le fait, qu'à notre connais-

sance, il n'existe pas de proposition qui satisfasse conjointement les trois critères caractérisant un résumé. En particulier, les approches permettant d'extraire ces motifs fréquents sur des fenêtres du flot ont l'avantage de restituer l'évolution du flot mais représente un résumé dont la taille est en constante hausse violant ainsi le critère de compacité.

Dans le chapitre 4, nous montrons comment la structure de graphe utilisée pour résumer les données brutes du flot peut être adaptée pour résumer l'historique des relations fréquentes entre éléments du flot (*i.e.* itemsets). Similairement au résumé de données brutes, l'historique des itemsets *rarement fréquents* est conservée plus durablement.

Ensuite, la séquentialité inhérente aux données d'un flot rend l'extraction de séquences fréquentes particulièrement pertinente. Malheureusement, une difficulté associée aux données multidimensionnelles et multiniveaux est que l'extraction de séquences exploitant ces spécificités est un processus difficile à cause de la taille de l'espace de recherche à parcourir. Nous nous sommes intéressés à cette problématique et avons proposé, au début du chapitre 5, *SALINES*, un algorithme qui permet, du fait d'une meilleure exploitation des hiérarchies, de découvrir des séquences ne pouvant être extraites par les approches existantes. Nous adaptons ensuite certains mécanismes issus de *SALINES* au contexte dynamique et proposons, toujours dans le chapitre 5, *ALIZES* une approche de résumé basée sur des séquences multidimensionnelles multiniveaux et adaptée au contexte des flots.

Prise en compte de la connaissance experte dans le processus de généralisation

La généralisation d'attributs numériques est bien souvent contextuelle dans le sens où une valeur numérique n'est souvent pas la seule information à considérer pour réaliser sa généralisation. Plus précisément, une connaissance experte est bien souvent nécessaire pour donner une étiquette sémantique correcte à un attribut numérique. A notre connaissance, aucun type de hiérarchies n'a été défini pour intégrer cette connaissance dans le processus de généralisation. Aussi, dans le chapitre 6, nous définissons formellement les hiérarchies contextuelles et proposons également la modélisation conceptuelle d'un entrepôt intégrant ces hiérarchies. En outre, nous proposons un modèle logique d'un tel entrepôt contextuel basé sur une modélisation relationnelle (*ROLAP*).

Perspectives

Les travaux effectués dans cette thèse peuvent se poursuivre de différentes façons. Quelques perspectives associées à chaque proposition ont d'ailleurs été brièvement présentées dans les sections de discussion des chapitres associés. Nous détaillons ici quelques perspectives qui nous tiennent à cœur. Nous discutons des perspectives à court terme visant à améliorer, soit les performances de construction des résumés, soit la qualité des connaissances qui y sont stockées. Enfin, nous distinguons quelques perspectives à long terme qui orientent nos travaux vers d'autres contextes d'application ou d'autres domaines de recherche.

Améliorer les performances de construction des résumés

Les perspectives associées à l'amélioration des techniques proposées dans cette thèse peuvent être divisées en deux catégories : celles visant à réduire la volumétrie et/ou la complexité des données en entrée et celles basées sur des optimisations structurelles et algorithmiques.

Réduire la complexité et/ou la volumétrie des données

Les techniques d'échantillonnages de flot de données sont désormais bien maîtrisées. Ainsi, il serait intéressant de réduire le volume de données à résumer grâce à l'utilisation de techniques d'échantillonnage. Cette perspective est également envisageable pour résumer un flot via l'utilisation de motifs fréquents car certaines approches existantes [RP07] garantissent formellement un taux d'erreur faible entre l'ensemble des motifs extrait avec ou sans échantillonnage.

Au cours des différentes expérimentations menées, nous avons vu que le passage à l'échelle des approches proposées était remis en cause lorsque le nombre de dimensions était trop important. Il serait donc intéressant de travailler sur la réduction du nombre de dimensions à considérer en recherchant, par exemple, l'existence de corrélations entre les dimensions.

Optimisations structurelles et algorithmiques

Dans certaines approches proposées dans ce travail, il pourrait être judicieux de s'interroger sur l'utilisation de nouvelles structures pour accroître les performances obtenues. Naturellement, l'utilisation de nouvelles structures imposerait de modifier les algorithmes proposés en conséquence. Typiquement, l'approche *SALINES* procède en deux étapes. La première (*i.e.* construction d'un automate des sous-séquences) est particulièrement critique quand il s'agit de fouiller une base de données volumineuse. Nous sommes malgré tout convaincu de l'utilité d'extraire de tels motifs et souhaitons investiguer de nouvelles pistes pour découvrir de telles séquences. Intuitivement,

l'utilisation du paradigme *PATTERN – GROWTH* [PHMa⁺01] et la définition de nouvelles techniques de projection de bases de données pourraient être utiles.

Améliorer la qualité des résumés produits

Parmi les perspectives associées à cette catégorie, l'utilisation des hiérarchies contextuelles dans les approches de construction de résumés ou d'extraction de séquences fréquentes pourrait être particulièrement intéressante. Typiquement, dans un contexte de résumés de données de capteurs, la création d'un résumé basé sur les généralisations sémantiques correctes des valeurs mesurées améliorerait son exploitation en permettant, par exemple, de repérer plus facilement les anomalies.

Mis à part dans le chapitre 6, toutes les approches proposées dans le cadre de cette thèse ont considérées que les hiérarchies des données étaient des arbres (*i.e.* hiérarchies simples). Or, comme nous l'avons vu dans l'état de l'art des catégories de hiérarchies existantes présenté dans le chapitre 6, il existe de nombreuses autres catégories de hiérarchies. L'adaptation de nos approches à ces catégories pourrait engendrer un gain sémantique dans les résumés produits mais ne serait pas exempte de difficultés. Par exemple, l'utilisation de hiérarchies floues permettrait la construction de résumés plus représentatifs mais il faudrait alors revoir la correction des mécanismes d'agrégation proposés dans les chapitres 3 et 4.

Enfin, concernant les approches basées sur les motifs fréquents (chapitres 4 et 5), il est regrettable que les motifs extraits n'exploitent pas les caractéristiques multidimensionnelles et hiérarchiques des données. A court terme, il s'agirait de rechercher dans la littérature des approches exploitant ces spécificités pour la découverte de motifs fréquents dans les flots.

Vers l'exploration de thèmes de recherche connexes

Comme nous l'avons évoqué dans le chapitre 4, il existe de nombreuses similarités entre le processus de mémorisation de l'information par un individu et les actions accomplies pour résumer un flot de données multidimensionnelles et multiniveaux. Aussi, s'inspirant de la théorie de la mémoire constructive, nous avons proposé, dans le chapitre 4, de ne retenir que les corrélations fréquentes entre éléments du flot. Forts de cette expérience, nous sommes convaincus que l'adaptation d'autres théories issues de la psychologie cognitive pourrait constituer des résumés de qualité. Par exemple, la *single-trace theory* rompt avec l'idée que la mémoire humaine est divisée en deux parties séparées (mémoire à court terme et à long terme) et propose une fonction mathématique pour déterminer la force d'un souvenir en fonction de son âge. Plusieurs paramètres permettent d'ajuster en temps réel cette force. Par exemple, l'évocation d'un souvenir augmente la valeur d'un

paramètre et a pour conséquence de consolider ce souvenir. De nombreux mécanismes décrits dans cette théorie semblent directement applicables dans notre contexte. On peut par exemple imaginer qu'un élément de l'historique serait conservé plus ou moins longtemps en fonction du nombre de fois où il est interrogé.

Ces dernières années, un autre domaine de la recherche informatique s'intéresse à la modélisation des comportements humains : les systèmes multi agent. De plus, il semble que la communauté de fouille de données s'intéresse depuis peu à l'utilisation de ces systèmes dans un contexte d'extraction de connaissances (*agent mining*). Il serait alors être intéressant d'étudier l'utilisation des agents dans un processus de résumé de flot de données. Par exemple, une colonie de fourmis pourrait parcourir la structure de graphe proposée dans le chapitre 4 pour consolider certaines parties de l'historique.

Vers l'exploitation des techniques de résumé

Dans cette section, nous développons quelques perspectives associées à l'exploitation des résumés de flot. En effet, nous pensons que leur simple consultation est très utile mais limitative. Similairement à l'émergence des techniques d'extraction de connaissances dans les bases de données, la fouille de résumés permettrait d'exhiber des connaissances nouvelles sur les flots de données. Par exemple, nous sommes convaincus que la découverte de périodicité et d'éléments atypiques est permise par l'approche proposée dans le chapitre 4. Il serait alors intéressant de pousser dans cette voie et de proposer de telles approches. De plus, les connaissances extraites (*e.g.* périodicité) sur le résumé en cours de construction pourrait enrichir la manière de résumer le flot.

Enfin, s'il est évidemment nécessaire que les résumés produits soient compréhensibles par un utilisateur humain, on peut s'interroger sur l'utilité d'être interprétable par un utilisateur non humain. Par exemple, la robotique est en pleine expansion. Typiquement, un robot évoluant dans un environnement inconnu intègre toutes les informations qu'il peut recueillir et les exploite pour s'adapter rapidement et efficacement à ce nouvel environnement. Intuitivement, bien que de nature hétérogène, les données recueillies par un robot peuvent être assimilées à un flot de données. Dès lors, il serait pertinent d'étudier à quel point une approche de résumé de flot peut améliorer les comportements de robots.

Bibliographie

- [ABB⁺04] A. ARASU, B. BABCOCK, S. BABU, J. CIESLEWICZ, M. DATAR, K. ITO, R. MOTWANI, U. SRIVASTAVA et J. WIDOM : Stream : The stanford data stream management system. *a book on data stream management edited by Garofalakis, Gehrke, and Rastogi*, 2004.
- [ACc⁺03] D. J. ABADI, D. CARNEY, U. ÇETINTEMEL, M. CHERNIACK, C. CONVEY, S. LEE, M. STONEBRAKER, N. TATBUL et S. ZDONIK : Aurora : a new model and architecture for data stream management. *The VLDB Journal*, 12(2):120–139, 2003.
- [Agg07] C.C. AGGARWAL : *Data streams : models and algorithms*. Springer-Verlag New York Inc, 2007.
- [AGP00] S. ACHARYA, P. B. GIBBONS et V. POOSALA : Congressional samples for approximate answering of group-by queries. *In SIGMOD '00 : Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 487–498, New York, NY, USA, 2000. ACM.
- [AGS97] R. AGRAWAL, A. GUPTA et S. SARAWAGI : Modeling multidimensional databases. *In Data Engineering, 1997. Proceedings. 13th International Conference on*, pages 232–243. IEEE, 1997.
- [AHWY03] C.C. AGGARWAL, J. HAN, J. WANG et P.S. YU : A framework for clustering evolving data streams. *In Proceedings of the 29th VLDB Conference*, Berlin, Germany, 2003.
- [AHWY04] C. C. AGGARWAL, J. HAN, J. WANG et P. S. YU : On demand classification of data streams. *In KDD*, pages 503–508, 2004.
- [AP03] R. A. ANGRYK et F. E. PETRY : Consistent fuzzy concept hierarchies for attribute generalization. *In Proceeding of the IASTED International Conference on Information and Knowledge Sharing (IKS)*, 2003.
- [AS94] R. AGRAWAL et R. SRIKANT : Fast algorithms for mining association rules in large databases. *In VLDB '94 : Proceedings of the 20th International Conference on Very Large Data Bases*, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

- [ASS06] A. ABELLÓ, J. SAMOS et F. SALTOR : YAM2 : a multidimensional conceptual model extending UML. *Information Systems*, 31(6):541–567, 2006.
- [AW00] Rakesh AGRAWAL et Edward L. WIMMERS : A framework for expressing and combining preferences. *SIGMOD Rec.*, 29:297–306, May 2000.
- [BCC⁺01] A BONIFATI, F CATTANEO, S CERI, A FUGGETTA et S PARABOSCHI : Designing Data Marts for Data Warehouses. *ACM Transactions on Software Engineering and Methodology*, 10(4):452–483, 2001.
- [BCF⁺05] D. BURDICK, M. CALIMLIM, J. FLANNICK, J. GEHRKE et T. YIU : Mafia : A maximal frequent itemset algorithm. *Knowledge and Data Engineering, IEEE Transactions on*, 17(11):1490–1504, 2005.
- [BGS01] P. BONNET, J. GEHRKE et P. SESHADRI : Towards sensor database systems. In *MDM '01 : Proceedings of the Second International Conference on Mobile Data Management*, pages 3–14, London, UK, 2001. Springer-Verlag.
- [BM98] A.G. B
"UCHNER et M.D. MULVENNA : Discovering internet marketing intelligence through online analytical web usage mining. *ACM Sigmod Record*, 27(4):54–61, 1998.
- [BR99] K. BEYER et Raghu RAMAKRISHNAN : Bottom-up computation of sparse and iceberg cube. *SIGMOD Rec.*, 28(2):359–370, 1999.
- [Bre84] L. BREIMAN : *Classification and regression trees*. Chapman & Hall/CRC, 1984.
- [BS97] D. BARBARÁ et M. SULLIVAN : Quasi-cubes : exploiting approximations in multidimensional databases. *SIGMOD Rec.*, 26(3):12–17, 1997.
- [BSSJ98] R BLIUJUTE, S SALTENIS, G SLIVINSKAS et C JENSEN : Systematic Change Management in Dimensional Data Warehousing. In *IIIrd International Baltic Workshop on Databases and Information Systems, Riga, Latvia*, pages 27–41, 1998.
- [BvE00] M BOEHNLEIN et A Ulbrich vom ENDE : Business Process Oriented Development of Data Warehouse Structures. In *Data Warehousing 2000 - Methoden Anwendungen, Friedrichshafen, Germany*. Physica Verlag, 2000.
- [CÇC⁺02] D. CARNEY, U. ÇETINTEMEL, M. CHERNIACK, C. CONVEY, S. LEE, G. SEIDMAN, M. STONEBRAKER, N. TATBUL et S. ZDONIK : Monitoring streams : A new class of data management applications. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 215–226. VLDB Endowment, 2002.
- [CCD⁺03] S. CHANDRASEKARAN, O. COOPER, A. DESHPANDE, M. J. FRANKLIN, J. M. HELLERSTEIN, W. HONG, S. KRISHNAMURTHY, S. MADDEN, V. RAMAN, F. REISS et M. A. SHAH : Telegraphcq : Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

- [CDTW00] J. CHEN, D.J. DEWITT, F. TIAN et Y. WANG : NiagaraCQ : A scalable continuous query system for internet databases. *ACM SIGMOD Record*, 29(2):379–390, 2000.
- [CF02] K.P. CHAN et A.W.C. FU : Efficient time series matching by wavelets. *In Data Engineering, 1999. Proceedings., 15th International Conference on*, pages 126–133. IEEE, 2002.
- [CFP⁺04] C. CORTES, K. FISHER, D. PREGIBON, A. ROGERS et F. SMITH : Hancock : A language for analyzing transactional data streams. *ACM Trans. Program. Lang. Syst.*, 26(2):301–338, 2004.
- [CJSS03] C. D. CRANOR, T. JOHNSON, O. SPATSCHECK et V. SHKAPENYUK : Gigascope : A stream database for network applications. *In Alon Y. HALEVY, Zachary G. IVES et AnHai DOAN, éditeurs : SIGMOD Conference*, pages 647–651. ACM, 2003.
- [CL03] J. H. CHANG et W. S. LEE : Finding recent frequent itemsets adaptively over on-line data streams. *In Proceedings of 9th International ACM SIGKDD international conference on Knowledge Discovery and Data Mining*, pages 487–492, New York, NY, USA, 2003.
- [Cod70] E.F. CODD : A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.
- [CT98] L. CABIBBO et R. TORLONE : A logical approach to multidimensional databases. *Advances in Database Technology—EDBT’98*, page 183, 1998.
- [CWYL09] L. CHANG, T. WANG, D. YANG et H. LUAN : Seqstream : Mining closed sequential patterns over stream sliding windows. *In Data Mining, 2008. ICDM’08. Eighth IEEE International Conference on*, pages 83–92. IEEE, 2009.
- [CWZ05] G. CHEN, X. WU et X. ZHU : Sequential pattern mining in multiple streams. *In ICDM ’05 : Proceedings of the Fifth IEEE International Conference on Data Mining*, pages 585–588, Washington, DC, USA, 2005. IEEE Computer Society.
- [esp] EsperTech - products - esper. <http://www.espertech.com/products/esper.php>.
- [EV01] M Minuto ESPIL et A A VAISMAN : Efficient Intensional Redefinition of Aggregation Hierarchies in Multidimensional Databases. *In IVth ACM International Workshop on Data Warehousing and OLAP (DOLAP 01), Atlanta, Georgia, USA*, pages 1–8. ACM Press, 2001.
- [FBB07] C FAVRE, F BENTAYEB et O BOUSSAID : Evolution et personnalisation des analyses dans les entrepôts de données : une approche orientée utilisateur. *In XXVème congrès INformatique des ORganisations et Systèmes d’Information et de DÉcision (INFORSID 07), Perros-Guirec, France*, pages 308 – 323, 2007.

- [FRM94] C. FALOUTSOS, M. RANGANATHAN et Y. MANOLOPOULOS : Fast subsequence matching in time-series databases. *ACM SIGMOD Record*, 23(2):419–429, 1994.
- [GBLP96] J. GRAY, A. BOSWORTH, A. LAYMAN et H. PIRAHESH : Data cube : A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In *ICDE '96 : Proceedings of the Twelfth International Conference on Data Engineering*, pages 152–159, Washington, DC, USA, 1996. IEEE Computer Society.
- [GCB⁺97] J. GRAY, S. CHAUDHURI, A. BOSWORTH, A. LAYMAN, D. REICHART, M. VENKATRAO, F. PELLOW et H. PIRAHESH : Data cube : A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.
- [GG07] J. GAMA et M.M. GABER : *Learning from data streams*. Springer-Verlag Berlin Heidelberg, 2007.
- [GHP⁺02] C. GIANNELLA, J. HAN, J. PEI, X. YAN et P. YU : Mining frequent patterns in data streams at multiple time granularities, 2002.
- [GKTD00] D. GUNOPULOS, G. KOLLIOS, V. J. TSOTRAS et C. DOMENICONI : Approximating multi-dimensional aggregate range queries over real attributes. *SIGMOD Rec.*, 29(2):463–474, 2000.
- [GMR98a] M. GOLFARELLI, D. MAIO et S. RIZZI : Conceptual Design of Data Warehouses from E/R Schemes. In *XXXIst Annual Hawaii International Conference on System Sciences (HICSS 98)*, Big Island, Hawaii, USA, volume 7, pages 334–343, 1998.
- [GMR98b] M. GOLFARELLI, D. MAIO et S. RIZZI : The dimensional fact model : a conceptual model for data warehouses. *International Journal of Cooperative Information Systems*, 2(3):215–247, 1998.
- [GZ02] K. GOUDA et M.J. ZAKI : Efficiently mining maximal frequent itemsets. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 163–170. IEEE, 2002.
- [GZK05] M.M. GABER, A. ZASLAVSKY et S. KRISHNASWAMY : Mining data streams : a review. *ACM Sigmod Record*, 34(2):18–26, 2005.
- [HAMS97] C.T. HO, R. AGRAWAL, N. MEGIDDO et R. SRIKANT : Range queries in OLAP data cubes. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 73–88. ACM, 1997.
- [HCD⁺05] J. HAN, Y. CHEN, G. DONG, J. PEI, B. W. WAH, J. WANG et Y. D. CAI : Stream cube : An architecture for multi-dimensional analysis of data streams. *Distributed and Parallel Databases*, 18(2):173–197, 2005.

- [HCY05] M.J. HSIEH, M.S. CHEN et P.S. YU : Integrating DCT and DWT for approximating cube streams. *In Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 179–186. ACM, 2005.
- [HMA⁺04] MA HAMMAD, MF MOKBEL, MH ALI, W.G. AREF, AC CATLIN, AK ELMAGARMID, M. ELTABAKH, MG ELFEKY, TM GHANEM, R. GWADERA *et al.* : Nile : A query processing engine for data streams. *In Data Engineering, 2004. Proceedings. 20th International Conference on*, page 851. IEEE, 2004.
- [HPDW01] J. HAN, J. PEI, G. DONG et K. WANG : Efficient computation of iceberg cubes with complex measures. *SIGMOD Rec.*, 30(2):1–12, 2001.
- [Inm96] W. H. INMON : *Building the data warehouse (2nd ed.)*. John Wiley & Sons, Inc., New York, NY, USA, 1996.
- [JJY00] H. JIAWEI, P. JIAN et Y. YIWEN : Mining frequent patterns without candidate generation. *In Proc of the ACM SIGMOD International Conference on Management of Data. Dallas, USA*, pages 1–12, 2000.
- [Kim96] R. KIMBALL : *The data warehouse toolkit : practical techniques for building dimensional data warehouses*. John Wiley & Sons, Inc. New York, NY, USA, 1996.
- [Lau03] A. LAURENT : Querying fuzzy multidimensional databases : Unary operators and their properties. *International Journal IJUFKS*, 11:31–45, 2003.
- [LBMS02] B LIST, R BRUCKNER, K MACHACZEK et J SCHIEFER : A Comparison of Data Warehouse Development Methodologies Case Study of the Process Warehouse. *In XIIIth International Conference on Database and Expert Systems Applications (DEXA 02), Aix-en-Provence, France*, volume 2453 de LNCS, pages 203–215. Springer, 2002.
- [LLS04] H.-F. LI, S.Y. LEE et M.-K. SHAN : An efficient algorithm for mining frequent itemsets over the entire history of data streams. *In Proceedings of 1st International Workshop on Knowledge Discovery in Data Streams*, Pisa, Italy, 2004.
- [LPH02] L.V.S. LAKSHMANAN, J. PEI et J. HAN : Quotient cube : How to summarize the semantics of a data cube. *In Proceedings of the 28th international conference on Very Large Data Bases*, pages 778–789. VLDB Endowment, 2002.
- [LPT00] L. LIU, C. PU et W. TANG : WebCQ-detecting and delivering information changes on the web. *In Proceedings of the ninth international conference on Information and knowledge management*, pages 512–519. ACM, 2000.
- [LPT02] L. LIU, C. PU et W. TANG : Continual queries for internet scale event-driven information delivery. *Knowledge and Data Engineering, IEEE Transactions on*, 11(4):610–628, 2002.

- [MFHH05] S.R. MADDEN, M.J. FRANKLIN, J.M. HELLERSTEIN et W. HONG : TinyDB : an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems (TODS)*, 30(1):122–173, 2005.
- [MI06] K. MORFONIOS et Y. IOANNIDIS : Cure for cubes : cubing using a rolap engine. *In VLDB '06 : Proceedings of the 32nd international conference on Very large data bases*, pages 379–390. VLDB Endowment, 2006.
- [Mil56] G. MILLER : Human memory and the storage of information. *Information Theory, IEEE Transactions on*, 2(3):129–137, 1956.
- [MM06] A. MARASCU et F. MASSEGLIA : Mining sequential patterns from data streams : a centroid approach. *J. Intell. Inf. Syst.*, 27(3):291–307, 2006.
- [MPT00] F. MASSEGLIA, P. PONCELET et M. TEISSEIRE : Web usage mining : How to efficiently manage new transactions and new clients. *Principles of Data Mining and Knowledge Discovery*, pages 179–204, 2000.
- [MS04] S. MUTHUKRISHNAN et M. STRAUSS : Approximate histogram and wavelet summaries of streaming data. Rapport technique, DIMACS TR, 2004.
- [Mut05] S. MUTHUKRISHNAN : *Data streams : Algorithms and applications*. Now Publishers Inc, 2005.
- [MV00] A O MENDELZON et A A VAISMAN : Temporal Queries in OLAP. *In XXVth International Conference on Very Large Data Bases (VLDB 00), Cairo, Egypt*, pages 242–253. Morgan Kaufmann, 2000.
- [MW03] T MORZY et R WREMBEL : Modeling a Multiversion Data Warehouse : A Formal Approach. *In Vth International Conference on Enterprise Information Systems (ICEIS 03), Angers, France*, pages 120–127, 2003.
- [MZ04] E. MALINOWSKI et E. ZIMÁNYI : OLAP hierarchies : A conceptual perspective. *In Advanced Information Systems Engineering*, pages 19–35. Springer, 2004.
- [MZ06] E. MALINOWSKI et E. ZIMÁNYI : Hierarchies in a multidimensional model : From conceptual modeling to logical representation. *Data & Knowledge Engineering*, 59(2):348–377, 2006.
- [PBTL99] N. PASQUIER, Y. BASTIDE, R. TAOUIL et L. LAKHAL : Discovering frequent closed itemsets for association rules. *Database Theory—ICDT'99*, pages 398–416, 1999.
- [PD02] C PHIPPS et K C DAVIS : Automating Data Warehouse Conceptual Schema Design and Evaluation. *In IVth International Workshop on Design and Management of Data Warehouses (DMDW 02), Toronto, Canada*, volume 58 de *CEUR Workshop Proceedings*, pages 23–32. CEUR-WS.org, 2002.

- [PG99] V. POOSALA et V. GANTI : Fast approximate answers to aggregate queries on a data cube. *In SSDBM '99 : Proceedings of the 11th International Conference on Scientific and Statistical Database Management*, pages 24–33, Washington, DC, USA, 1999. IEEE Computer Society.
- [PHM⁺04] J. PEI, J. HAN, B. MORTAZAVI-ASL, J. WANG, H. PINTO, Q. CHEN, U. DAYAL et M. C. HSU : Mining sequential patterns by pattern-growth : The prefixspan approach. *IEEE Transactions on Knowledge and Data Engineering*, pages 1424–1440, 2004.
- [PHMa⁺01] J. PEI, J. HAN, B. MORTAZAVI-ASL, H. PINTO, Q. CHEN et U. DAYAL : Prefixspan : Mining sequential patterns efficiently by prefix-projected pattern growth. *In Proceedings of 17th International Conference on Data Engineering (ICDE 01)*, pages 215–224, Heidelberg, Germany, 2001.
- [PIR03] V PERALTA, A ILLARZE et R RUGGIA : On the Applicability of Rules to Automate Data Warehouse Logical Design. *In Ist International Workshop on Decision Systems Engineering (DSE 03), in conjunction with the XVth International Conference on Advanced Information Systems Engineering (CAiSE 03), Klagenfurt/Velden, Austria*, volume 75 de *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.
- [PJ99] T.B. PEDERSEN et C.S. JENSEN : Multidimensional data modeling for complex data. *In Data Engineering, 1999. Proceedings., 15th International Conference on*, pages 336–345. IEEE, 1999.
- [PJD99] T. B. PEDERSEN, C. S. JENSEN et C. E. DYRESON : Supporting imprecision in multidimensional databases using granularities. *In Proceedings of SSDBM'99*, page 90. IEEE Computer Society, 1999.
- [PLT06] M. PLANTEVIT, A. LAURENT et M. TEISSEIRE : HYPE : mining hierarchical sequential patterns. *In Proceedings of the 9th ACM international workshop on Data warehousing and OLAP*, pages 19–26. ACM New York, NY, USA, 2006.
- [PLT07] M. PLANTEVIT, A. LAURENT et M. TEISSEIRE : Extraction d'Outliers dans des Cube de Données : une Aide à la Navigation. *In EDA'07 : Entrepôts de Données et Analyse en ligne*, pages 113–130, 2007.
- [PLT08] M. PLANTEVIT, A. LAURENT et M. TEISSEIRE : Up and down : Mining multidimensional sequential patterns using hierarchies. *In Proceedings of the 10th international conference on Data Warehousing and Knowledge Discovery*, pages 156–165. Springer-Verlag Berlin, Heidelberg, 2008.
- [PM02] I. POPIVANOV et R.J. MILLER : Similarity search over time-series data using wavelets. *In Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 212–221. IEEE, 2002.

- [PMC05] J. PEI, M. CHO et D. Wai-Lok CHEUNG : Cross table cubing : Mining iceberg cubes from data warehouses. *In SDM*, 2005.
- [Poe96] V POE : *Building a Data Warehouse for Decision Support*. Prentice Hall, 1996.
- [PSB92] A.D. PREECE, R. SHINGHAL et A. BATAREKH : Principles and practice in verifying rule-based systems. *The Knowledge Engineering Review*, 7(02):115–141, 1992.
- [PSV⁺07] N. POLYZOTIS, S. SKIADOPOULOS, P. VASSILIADIS, A. SIMITSIS et N.E. FRANTZELL : Supporting streaming updates in an active data warehouse. *In Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 476–485. IEEE, 2007.
- [Qui93] J.R. QUINLAN : *C4. 5 : programs for machine learning*. Morgan Kaufmann, 1993.
- [RALT06] S. RIZZI, A. ABELLÓ, J. LECHTENB
"ORGER et J. TRUJILLO : Research in data warehouse modeling and design : dead or alive? *In Proceedings of the 9th ACM international workshop on Data warehousing and OLAP*, pages 3–10. ACM, 2006.
- [RM97] D. RAFIEI et A. MENDELZON : Similarity-based queries for time series data. *In Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 13–25. ACM, 1997.
- [RP07] Chedy RAÏSSI et Pascal PONCELET : Sampling For Sequential Pattern Mining : From Static Databases to Data Streams. *In ICDM'07 : International Conference on Data Mining*, page 6. IEEE, 10 2007.
- [RP08] C. RAISSI et M. PLANTEVIT : Mining multidimensional sequential patterns over data streams. *In DaWak*, 2008.
- [RPT06] C. RAISSI, P. PONCELET et M. TEISSEIRE : Speed : Mining maximal sequential patterns over data streams. *In Proceedings of the 3rd IEEE International Conference on Intelligent Systems (IEEE IS 2006)*, London, UK, 2006.
- [RTT07] F. RAVAT, O. TESTE et R. TOURNIER : OLAP aggregation function for textual data warehouse. *In Ninth International Conference on Enterprise Information Systems (ICEIS), vol. DISI, INSTICC Press*, pages 151–156, 2007.
- [RTTZ08] F. RAVAT, O. TESTE, R. TOURNIER et G. ZURFLUH : Algebraic and graphic languages for OLAP manipulations. *International Journal of Data Warehousing and Mining*, 4(1):17–46, 2008.
- [RTZ01] F RAVAT, O TESTE et G ZURFLUH : Modélisation multidimensionnelle des systèmes décisionnels. *In EGC*, pages 201–212, 2001.

- [SA07] D. L. SCHACTER et D. R. ADDIS : The cognitive neuroscience of constructive memory : remembering the past and imagining the future. *Philos Trans R Soc Lond B Biol Sci*, 362(1481):773–786, May 2007.
- [SBHD04] C. SAPIA, M. BLASCHKA, G. H. OFLING et B. DINTER : Extending the E/R model for the multidimensional paradigm. *Advances in Database Technologies*, pages 1947–1947, 2004.
- [SBW08] J. SRIBUABAN, V. BOONJING et J. WERAPUN : Frequent closed multi-dimensional multi-level pattern mining. In *Proceedings of the Fourth IASTED International Conference on Advances in Computer Science and Technology*, pages 201–206. ACTA Press, 2008.
- [SCDT00] J. SRIVASTAVA, R. COOLEY, M. DESHPANDE et P.N. TAN : Web usage mining : Discovery and applications of usage patterns from web data. *ACM SIGKDD Explorations Newsletter*, 1(2):12–23, 2000.
- [SFG05] A SOUSSI, J FEKI et F GARGOURI : Approche semi-automatisée de conception de schémas multidimensionnels valides. In *1ère journée sur les Entrepôts de Données et l'Analyse en ligne (EDA 05), Lyon*, volume B-1 de *Revue des Nouvelles Technologies de l'Information*, pages 71–90. Cépaduès Editions, 2005.
- [SH98] M. SULLIVAN et A. HEYBEY : Tribeca : A system for managing large databases of network traffic. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, page 2. USENIX Association, 1998.
- [SHX04] Z. SHAO, J. HAN et D. XIN : Mm-cubing : Computing iceberg cubes by factorizing the lattice space. In *SSDBM*, pages 213–222, 2004.
- [SSS82] M. SUWA, A.C. SCOTT et E.H. SHORTLIFFE : An approach to verifying completeness and consistency in a rule-based expert system. *AI Magazine*, 3(4):16, 1982.
- [TGIK02] N. THAPER, S. GUHA, P. INDYK et N. KOUDAS : Dynamic multidimensional histograms. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 428–439. ACM, 2002.
- [Tor03] R. TORLONE : Multidimensional Models. *Multidimensional databases : problems and solutions*, page 69, 2003.
- [Tou07] R. TOURNIER : *Analyse en ligne (OLAP) de documents*. Thèse de doctorat, Université Paul Sabatier - Toulouse III, 2007.
- [VSMP09] L. VINCESLAS, JE SYMPHOR, A. MANCHERON et P. PONCELET : SPAMS : Une nouvelle approche incrémentale pour l'extraction de motifs séquentiels fréquents dans les data streams. *EGC, Volume RNTI-E-15 of Revue des Nouvelles Technologies de l'Information*, pages 205–216, 2009.

- [VW99] J.S. VITTER et M. WANG : Approximate computation of multidimensional aggregates of sparse data using wavelets. *In Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pages 193–204. ACM, 1999.
- [VWI98] J.S. VITTER, M. WANG et B. IYER : Data cube approximation and histograms via wavelets. *In Proceedings of the seventh international conference on Information and knowledge management*, pages 96–104. ACM, 1998.
- [XHLW03] D. XIN, J. HAN, X. LI et B. W. WAH : Star-cubing : computing iceberg cubes by top-down and bottom-up integration. *In vldb'2003 : Proceedings of the 29th international conference on Very large data bases*, pages 476–487. VLDB Endowment, 2003.
- [XSHL06] D. XIN, Z. SHAO, J. HAN et H. LIU : C-cubing : Efficient computation of closed cubes by aggregation-based checking. *In ICDE*, page 4, 2006.
- [ZAD88] L. ZADEH : Fuzzy logic. *Computer*, 21:83–93, 1988.
- [ZDN97] Y. ZHAO, Prasad M. DESHPANDE et J. F. NAUGHTON : An array-based algorithm for simultaneous multidimensional aggregates. *SIGMOD Rec.*, 26(2):159–170, 1997.
- [ZRL06] T. ZHANG, R. RAMAKRISHNAN et M. LIVNY : Birch : An efficient data clustering method for very large databases. *In SIGMOD*, Montreal, Canada, June 2006.
- [ZS02] Y. ZHU et D. SHASHA : Statstream : Statistical monitoring of thousands of data streams in real time. *In Proceedings of the 28th international conference on Very Large Data Bases*, pages 358–369. VLDB Endowment, 2002.

Articles publiés dans le cadre de cette thèse

Conférences internationales

- Mining sequential patterns from MODIS time series for cultivated area mapping
Yoann Pitarch, Elodie Vontrou, Agnes Begue, Maguelonne Teisseire et Fadi Badra
Proc. of the 13th 4th AGILE International Conference on Geographic Information Science,
à paraître Utrecht, Pays Bas, Avril 2011

- Context-Aware Generalization for Cube Measures
Yoann Pitarch, Cécile Favre, Anne Laurent et Pascal Poncelet
Proc. of the 13th International Workshop On Data Warehousing and OLAP (DOLAP 2010),
colocated with CIKM 2010, pp 99-104, Toronto, Canada, 30 Octobre 2010

- Summarizing Multidimensional Data Streams : A Hierarchy-Graph-Based Approach
Yoann Pitarch, Anne Laurent et Pascal Poncelet
Proc. of the 14th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD
2010), pp 335-342, Hyderabad, Inde, 21-24 Juin 2010

- TIGER : Querying a Relational Table through Criteria Extension
Yoann Pitarch, Dominique Laurent, Pascal Poncelet et Nicolas Spyrtos
Proc. of the 2nd IEEE International Conference on Soft Computing and Pattern Recognition
(SoCPaR 2010), pp 142-149, Cergy-Pontoise, France, 7-10 Décembre 2010

- A Conceptual Model for Handling Personalized Hierarchies
Yoann Pitarch, Anne Laurent et Pascal Poncelet.
Proc. of the International ACM Conference on Management of Emergent Digital EcoSystems
(MEDES 2009), pp 96-100 Lyon, France, Octobre 2009

- Multidimensional Data Streams Summarization Using Extended Tilted-Time Windows
Yoann Pitarch, Anne Laurent, Marc Plantevit et Pascal Poncelet
Proc. of the 5th International Symposium on Frontiers of Information Systems and Network Applications (FINA 2009) colocated with AINA 2009), pp 250-254, Bradford, UK, Mai 2009
- A Hierarchy-Based Method for Synthesizing Frequent Itemsets Extracted from Temporal Windows
Yoann Pitarch, Anne Laurent et Pascal Poncelet
Proc. of the IEEE International Conference on Soft Computing and Pattern Recognition (SoCPaR 2009), pp 192-198 Malacca, Malaysia, Décembre 2009

Revues nationales

- Fenêtres sur cubes Yoann Pitarch, Anne Laurent, Marc Plantevit et Pascal Poncelet.
Article sélectionné lors de la conférence BDA 08 RSTI série ISI, Volume 15, pp. 9-33, Janvier 2010

Conférences nationales

- TIGER : interrogation d'une table relationnelle par extension de critères
Yoann Pitarch, Dominique Laurent, Pascal Poncelet et Nicolas Spyrtos
26eme journées des Bases de Données Avancées (BDA 2010), pp. 76-95 Toulouse, France, 19-22 Octobre 2010
- Analyse flexible dans les entrepôts de données : quand les contextes s'en mêlent
Yoann Pitarch, Cécile Favre, Anne Laurent et Pascal Poncelet
6èmes Journées francophones sur les Entrepôts de Données et l'Analyse en ligne (EDA 2010), pp. 60-74 Djerba, Tunisie, 11-13 Juin 2010
- SALINES : un automate au service de l'extraction de motifs séquentiels multidimensionnels
Yoann Pitarch, Lionel Vincelas, Anne Laurent, Pascal Poncelet et Jean Emile Symphor
Extraction et Gestion des Connaissances (EGC 2010), pp. 49-54 Hammamet, Tunisie, 27-29 Janvier 2010
- Résumé généraliste de flux de données
Collectif d'auteurs issus du projet MIDAS

Extraction et Gestion des Connaissances (EGC 2010), pp. 255-260 Hammamet, Tunisie, 27-29 Janvier 2010

- Une structure basée sur les hiérarchies pour synthétiser les itemsets fréquents extraits dans des fenêtres temporelles

Yoann Pitarch, Anne Laurent et Pascal Poncelet

Extraction et Gestion des Connaissances (EGC 2010), pp. 711-712 Hammamet, Tunisie, 27-29 Janvier 2010

- Fenêtre sur cube

Yoann Pitarch, Anne Laurent, Marc Plantevit et Pascal Poncelet

24ièmes Journées des Bases de Données Avancées (BDA 2008), pp 13-32, Guiulherand-Granges, France, octobre 2008

Annexe

Annexe A Survol des DSMS existants

Annexe A

Survol des DSMS existants

Dans cette annexe, nous dressons un panorama des DSMS (Data Stream Management Systems) existants et les divisons en deux catégories :

- Développement par des entités privées de DSMS spécifiques en réponse à un fort besoin opérationnel (AT&T avec GigaScope et Hancock par exemple) et développement par des entités universitaires de solutions répondant à un besoin spécifique qu'on classe rétrospectivement dans la catégorie des DSMS (StatStream, WebCQ par exemple) ;
- Solutions généralistes (Stream, TelegraphCQ, Aurora, Aleri, Coral8 . . .)

Si le passage en phase commerciale peut laisser à penser que le domaine de la gestion de flux est arrivé à maturité, l'exploration de domaines comme la gestion de flux d'événements complexes fait apparaître de belles perspectives, avec des applications toujours plus nombreuses et des développements formels toujours plus nécessaires. Nous parcourons sommairement ici l'écosystème des DSMS, en présentant des systèmes spécifiques et des systèmes génériques. Il ne s'agit aucunement de nous lancer dans une évaluation comparative de tous les systèmes.

A.1 DSMS spécifiques

Nous présentons ici une liste non exhaustive de DSMS spécifiques à un domaine d'application :

- COUGAR (Cornell)
- GigaScope (ATT)
- Hancock (ATT)
- NiagaraCQ (OGI/Wisconsin)
- OpenCQ (Georgia Tech)
- StatStream
- Streaminer (UIUC)
- TinyDB (Berkeley)

- Tribeca (Bellcore)
- WebCQ (Georgia Tech)

COUGAR [BGS01] Le projet Cougar, mené à l'Université de Cornell, vise à programmer les capteurs d'un réseau à l'aide de requêtes exprimées dans un langage déclaratif de haut niveau (variante de SQL). Le réseau est constitué de capteurs non mobiles connectés sans fil. Les contraintes portent sur la capacité de communication (bande passante), la consommation d'énergie et la capacité de calcul. Chaque capteur constitue une source de données structurées. Le réseau contient des nœuds de visualisation : les données des capteurs sont poussées vers ces nœuds dans lesquelles elles sont stockées jusqu'à ce qu'elles soient consommées par l'application. La problématique est de fournir une interface expressive de programmation à l'aide de requêtes tout en assurant une optimisation de la gestion des ressources. L'utilisateur est alors en mesure de modifier dynamiquement le comportement du réseau tout en le préservant au maximum en minimisant l'utilisation des ressources.

GigaScope [CJSS03] GigaScope est un système propriétaire (ATT) orienté gestion de réseaux : analyse de trafic, détection d'intrusion, analyse de configuration des routeurs, . . . Une particularité de GigaScope est l'abandon du modèle fenêtré explicite. L'évaluation d'opérateurs bloquants repose sur une analyse des champs ordonnés des flux et des propriétés de la requête.

Le langage proposé est GSQL, une restriction du langage SQL augmenté par des opérations spécifiques au traitement de flux. Ainsi, GSQL permet la spécification de sélections, de jointures, d'agrégations et de fusions de flux. Une jointure concerne deux flux uniquement et doit inclure une contrainte sur un attribut d'agencement. Pour une agrégation, l'un des champs de regroupement doit être un attribut d'agencement. L'opérateur de fusion assemble plusieurs flux en un en conservant la propriété d'agencement d'un champ. L'utilisateur peut développer ses propres fonctions.

Au-delà des aspects fonctionnels de GigaScope, les auteurs de [CJSS03] décrivent leur retour d'expérience suite à des tests d'évaluation de performance. Nous les reportons ici car ils nous paraissent avoir une portée générale :

- Réduire la charge d'une application est critique pour les performances et le plus en amont est le mieux ;
- Passer par le disque peut porter un coup fatal aux performances
- Un plan de requêtes suffisamment complexe nécessitera une approximation des résultats ainsi qu'un échantillonnage.

Ils ajoutent également, rejoignant [CÇC⁺02], que la mesure de performance d'un DSMS n'est pas la vitesse à laquelle il produit le résultat mais le niveau de charge qu'il peut soutenir avant de faire sauter des n-uplets. Notons aussi que, si l'on doit perdre des n-uplets, il est pertinent de

chercher à éliminer les n-uplets les moins intéressants. Là encore, les auteurs proposent un autre point de vue que celui donné dans [CÇC⁺02], plus simple à analyser et à implémenter : les n-uplets intéressants sont ceux produits le plus en aval.

Hancock [CFP⁺04] Hancock est un langage spécifique développé par AT&T afin de calculer des agrégats (ou *signatures*) à partir de flux de données transactionnelles. Basé sur du C, ce langage permet d'écrire et de lire des programmes de manière plus efficace (en temps et en mémoire).

NiagaraCQ [CDTW00] NiagaraCQ est un système de gestion de plans de requêtes continues sur des bases de données de l'internet. L'objectif est de gérer un grand nombre de requêtes, de manière dynamique (les requêtes sont ajoutées, éliminées continuellement). L'efficacité de la gestion est assurée par le groupement des requêtes similaires, pour un traitement non redondant. Le système est distribué et permet de construire des requêtes dans un langage proche de XML-QL sur des bases XML distribuées. Les requêtes sont écrites en XML-QL et se voient adjoindre une information temporelle : date de première exécution de la requête, date d'expiration de la requête et délai séparant deux exécutions de la requête.

OpenCQ [LPT02] OpenCQ permet la définition et le calcul de requêtes sur des bases de données persistantes structurées distribuées sur un large réseau (données bibliographiques). Les requêtes ont par exemple pour but de notifier un changement dans la base. WebCQ est une adaptation d'OpenCQ qui permet le suivi de pages web (données "non structurées"). Le langage de requête est un langage orienté objet.

StatStream [ZS02] StatStream est un outil de calcul en temps réel d'indicateurs statistiques (corrélation) sur un ensemble de séries temporelles. StatStream prend en entrée un ensemble de flux et détermine les paires de flux dont la corrélation dépasse un seuil fixé par l'utilisateur. La périodicité de calcul de l'indicateur est fixée par l'utilisateur et la taille de la fenêtre sur laquelle est calculé l'indicateur est un paramètre utilisateur. L'intérêt de StatStream repose sur sa capacité à traiter un grand nombre de flux.

TinyDB [MFHH05] TinyDB est un système d'extraction d'informations dans un réseau de capteurs. Il fournit une interface de type SQL afin de spécifier les requêtes d'extraction d'information, ainsi que des paramètres spécifiques comme le débit. Pour une requête particulière, TinyDB collecte les données issues des capteurs, les filtre, les agrège et les envoie à destination d'un PC, tout en assurant une exploitation efficace des ressources du réseau. TinyDB résulte d'un travail effectué à l'Université de Berkeley, terminé en 2003. Le logiciel est disponible gratuitement.

Tribeca [SH98] L'objectif de Tribeca est analogue à celui de GigaScope : l'analyse en ligne du trafic Internet. Le langage est un langage procédural (la requête est structurée comme un graphe acyclique orienté) définissant trois types d'opérations simples :

- Qualification : filtre les n-uplets d'un flux
- Projection : recombinaison des champs d'un flux
- Agrégation : application d'une fonction à tous les n-uplets d'un flux retournant une valeur unique,

et différents types d'opérations combinant plusieurs flux :

- multiplexage/démultiplexage : partition et recombinaison d'un flux, afin d'appliquer un traitement à chacune des parties
- fenêtrage : fenêtres glissantes ou sautantes - filtre sur fenêtres : comparaison de n-uplets proches dans le temps.

WebCQ ([LPT00]) WebCQ est un système de détection et de notification des changements de pages web arbitraires. C'est une application fonctionnant en tant que serveur. Elle est composée de plusieurs parties, dont les trois principales sont les suivantes :

- Un robot détectant les changements dans les pages web ;
- Un outil de présentation personnalisée des changements ;
- Un service de notification.

L'utilisateur enregistre sa requête (une sentinelle) à l'aide d'un formulaire HTML et spécifie le type d'information qu'il souhaite voir surveillée. La sentinelle est soumise au moteur dès lors que l'utilisateur a spécifié une adresse mail pour les notifications. Le moteur lance le robot périodiquement. Lorsque des changements intéressants sont détectés, un courriel de notification est envoyé. WebCQ résulte d'un travail effectué à Georgia Tech, terminé au début des années 2000. Le logiciel est disponible gratuitement.

A.2 DSMS généralistes

STREAM (université de Stanford) [ABB⁺04] STREAM est un DSMS implémenté en C++ et doté d'une interface graphique. Il s'appuie sur le langage de requête CQL, extension de SQL, afin de formuler des requêtes continues sur des flux de données. STREAM manipule des flux, des relations et leurs interactions. Ainsi, CQL établit :

- Un langage de requête relationnel, pour opérer sur les relations ;
- Un langage de spécification de fenêtre pour convertir des flux en relations,
- Un groupe de trois opérateurs pour transformer les relations en flux. STREAM est basé sur un temps logique et, lorsque la datation n'est pas explicite, le système passe automatiquement en datation implicite. STREAM permet l'usage de fenêtres glissantes uniquement.

De plus, il est impossible de définir des requêtes ad hoc. Il autorise l'approximation des résultats d'une requête pour faire face aux situations de surcharge.

TelegraphCQ /TruViso (université de Berkeley) [CCD⁺03] Après une première tentative infructueuse, qui consistait à implémenter un système de gestion en Java, l'équipe du projet TelegraphCQ s'est orientée vers le développement d'une extension du SGBD open source postgresSQL. Les résultats du travail initial sur le requêtage continu ont été réemployés, notamment les eddies (pour le routage des n-uplets et la planification des opérateurs) et les fjords (pour les communications inter-opérateurs). Le système TelegraphCQ est décrit plus en détail dans une prochaine section. L'annexe A contient les notes relatives à son utilisation. TruViso (initialement Aminsight) est la version commerciale de TelegraphCQ.

Aurora, Medusa, Borealis (universités de Brandeis et de Brown, MIT) [ACc⁺03] Le système Aurora, et ses congénères Medusa et Borealis, est décrit plus précisément dans une prochaine section. Sa caractéristique principale est de proposer une interface de développement d'application à l'aide de diagrammes (boîtes et flèches). StreamBase est la version commerciale d'Aurora.

Nile (université de Purdue) [HMA⁺04] Nile est un système d'administration de flux de données. Il présente des technologies de base de données avancées nécessaires pour diriger et traiter des flux de données en ligne. Quelques caractéristiques de Nile sont la prise en compte du paradigme flux-entrant/flux sortant, la gestion fenêtrées glissantes, ... Le projet semble actuellement à l'arrêt.

Esper [esp] Esper est un DSMS développé sous licence GNU (General Public License) par la société *EsperTech Inc* (Wayne , New Jersey). Ce DSMS est basé sur un moteur ESP et CEP (« Complex Event Processing ») capable de requêtes continues sur flux de données, d'agrégations et de jointures et d'extraire du flux les événements intéressants et l'information significative.

Résumé de Flots de Données : Motifs, Cubes et Hiérarchies

Avec le développement des TIC, le volume de données disponibles a explosé et a démocratisé les flots. Un flot de données peut être défini comme une séquence non bornée de données très précises et circulant à grande vitesse. Le stocker intégralement est par définition impossible motivant alors le besoin de proposer des techniques de résumé permettant d'analyser a posteriori cet historique. En outre, un grand nombre de flots de données présentent un caractère multidimensionnel et multiniveaux que très peu d'approches existantes exploitent. Ainsi, l'objectif de ces travaux est de proposer des méthodes de résumé exploitant ces spécificités multidimensionnelles et applicables dans un contexte dynamique. Nous nous intéressons à l'adaptation des techniques OLAP (On Line Analytical Processing) et plus particulièrement, à l'exploitation des hiérarchies de données pour réaliser cette tâche. Pour aborder cette problématique, nous avons mis en place trois angles d'attaque. Tout d'abord, après avoir discuté et mis en évidence le manque de solutions satisfaisantes, nous proposons deux approches permettant de construire un cube de données alimenté par un flot. Le deuxième angle d'attaque concerne le couplage des approches d'extractions de motifs fréquents (itemsets et séquences) et l'utilisation des hiérarchies pour produire un résumé conservant les tendances d'un flot. Enfin, les catégories de hiérarchies existantes ne permettent pas d'exploiter les connaissances expertes dans le processus de généralisation. Nous pallions ce manque en définissant une nouvelle catégorie de hiérarchies, dites contextuelles, et en proposant une modélisation conceptuelle, graphique et logique d'un entrepôt de données intégrant ces hiérarchies contextuelles. Cette thèse s'inscrivant dans un projet ANR (MIDAS), une plateforme de démonstration intégrant les principales approches de résumé a été mise au point. De plus, la présence de partenaires industriels dans le projet MIDAS (e.g., Orange Labs et EDF RD) a permis de valider nos approches sur des jeux de données réelles.

Mots-clés : Flots de données, Hiérarchies, Motifs fréquents, Entrepôts de données.

Discipline : Informatique

Laboratoire : Laboratoire d'Informatique de Robotique et de Micro-électronique de Montpellier
Université Montpellier II - CNRS (UMR 5506)
161 rue Ada - 34095 Montpellier Cedex 5 France