Construction et affinement de hiérarchies à partir de données semi-structurées

Nathalie Pernelle*, — Véronique Ventos* — Marie-Christine Rousset***

* *LRI, UMR 8623 CNRS - Université Paris-Sud,*
Bâtiment 490, 91405 Orsay cedex, France,
{pernelle,ventos,mcr}@lri.fr

** *Ecole d'ingénieurs EISTI*
Avenue du parc, 95000 Cergy-Pontoise

ABSTRACT. *In many applications, it becomes crucial to help users to access to a huge amount of data by clustering them in a small number of classes described at an appropriate level of abstraction. In this paper, we present an approach based on the use of two languages of description of classes for the automatic clustering of semistructured data. The first language of classes has a high power of abstraction and guides the construction of a basic lattice of classes covering the whole set of the data. The second language of classes, more expressive and more precise, is the basis for the refinement of a part of the lattice that the user wants to focus on.*

RÉSUMÉ. *Dans de nombreuses applications, il devient crucial de faciliter l'accès de l'utilisateur à des données trop nombreuses en les regroupant en un nombre plus petit de classes décrites à un niveau d'abstraction plus pertinent pour l'utilisateur. Dans cet article, nous présentons une approche caractérisée par l'utilisation de deux langages de classes pour la classification automatique de données semi-structurées. Le premier langage de classes, à fort pouvoir d'abstraction, guide la construction d'un treillis sommaire de classes couvrant l'ensemble des données. Le second langage de classes, plus expressif et plus précis, est utilisé dans une phase d'affinement d'une partie du treillis que l'utilisateur a pu pointer comme l'intéressant tout particulièrement.*

KEYWORDS: *conceptual classification, knowledge representation, language shifts*

MOTS-CLÉS : *classification conceptuelle, représentation des connaissances, changements de langages*

1. Introduction

Avec l'explosion du Web et du commerce électronique, les utilisateurs ont un accès potentiel à des données de plus en plus nombreuses. Dans ce contexte, le besoin devient crucial d'outils permettant d'organiser les données en classes décrivant les instances qu'elles regroupent de façon plus concise et à un niveau d'abstraction plus pertinent pour un utilisateur novice. Par exemple, de plus en plus d'entreprises ou de vendeurs veulent mettre à disposition sur le Web leurs produits dont la description est la plupart du temps disponible dans un certain format imposé par l'outil de stockage des données utilisé (SGBD, fichiers XML, etc). Dans le cadre du projet GAEL [1], nous nous intéressons à la construction de catalogues flexibles permettant aux utilisateurs un accès aux produits qui soit personnalisé et guidé par une hiérarchie de classes.

Le problème est, à partir d'un ensemble de données existantes, de créer de façon automatique un ensemble de classes regroupant des données en fonction de l'existence de propriétés communes, en associant à chaque classe une description permettant de rendre compte de ces propriétés communes. Toute la difficulté consiste à choisir un langage de classes présentant un bon compromis entre pouvoir d'expression et pouvoir d'abstraction. Un langage de classes avec un pouvoir d'abstraction élevé présente l'intérêt de limiter le nombre de classes construites à partir des données, la description de chaque classe capturant un nombre important de données. Un pouvoir d'abstraction élevé du langage de classes peut donc s'avérer nécessaire pour la mise en oeuvre d'algorithmes de construction de classes à partir de données de complexité raisonnable. Or, la contrepartie d'un pouvoir d'abstraction élevé est un pouvoir d'expression limité. Un langage de classes expressif est pourtant utile pour décrire de façon fine un sous-ensemble des données qui intéresse tout particulièrement l'utilisateur. Le risque d'un langage de classes trop expressif est de produire un nombre de classes construites à partir des données trop important pour que la hiérarchie complète des classes puisse être présentée à l'utilisateur.

La caractéristique de notre approche est de concilier pouvoir d'abstraction et pouvoir d'expression du langage de classes en procédant en deux étapes pour la construction de classes. La première étape, fondée sur un langage de classes simple, prend en entrée l'ensemble des données et construit un premier treillis [2] sommaire de classes. La seconde étape est une opération d'affinement d'une partie du treillis. Elle se focalise sur le sous-ensemble des données qui sont instances d'une classe du treillis pointée par l'utilisateur comme l'intéressant tout particulièrement. Pour ce sous-ensemble des

1. GAEL est un projet financé par le Ministère de l'Industrie dans le cadre du Réseau National de Recherche en Telecommunications (RNRT) et réalisé en partenariat avec l' équipe VERSO de l'INRIA et l'entreprise Matchvision.
2. Une hiérarchie sous forme de treillis permet aux classes d'avoir plusieurs caractérisations et donc d'introduire la notion de point de vue.

données, elle affine la classification de la première étape en se fondant sur un langage de classes plus expressif.

Notre approche s'applique sur des données semi-structurées au sens suivant : chaque donnée est décrite par un ensemble variable d'attributs monovalués ou multivalués, et est étiquetée par un type. Le type des données fournit une première structuration des données en classes (disjointes), que nous appelons classes de base. Ces données peuvent représenter les produits d'un catalogue. Notre approche permet alors de construire dans un premier temps un treillis de classes qui puisse être utilisé pour montrer à l'utilisateur du catalogue une première hiérarchie grossière ne comportant qu'un nombre peu élevé de classes, ces classes étant décrites par un ensemble d'informations concises mais représentatives de la classe. Ceci permet de présenter tout le catalogue à l'utilisateur sans le noyer par un trop grand nombre d'informations. De plus, cela permet dans le même temps de diminuer le coût de construction de cette première hiérarchie. Il s'agira ensuite, si l'utilisateur est interessé par un sous-ensemble de la hiérarchie, de l'affiner. Pour diminuer de manière pertinente la quantité d'information prise en compte dans la première hiérarchie, nous avons choisi de ne pas prendre en compte les valeurs des attributs présents dans la description des instances. Ces valeurs ne seront prises en compte que lors de la construction de la hiérarchie affinée.

Notre approche a été appliquée sur des données réelles provenant de C/Net. C/Net est un catalogue de produits informatiques dans lequel chaque produit est décrit par un ensemble d'attributs mono ou multivalués. Pour chaque produit, un attribut précise une classe unique auquel le produit appartient. Les données sont donc semi-structurées et étiquetées. C/Net comporte 2274 produits décrits à l'aide d'un total de 234 attributs et répartis dans 66 classes de base disjointes.

Le plan de notre article est le suivant. Dans la section 2, nous définissons formellement les différents langages qui sont utilisés dans notre approche : le langage des instances, le langage des classes de la hiérarchie grossière ainsi que le langage des classes de la hiérarchie affinée. La section 3 est consacrée à la présentation des étapes de construction et d'affinement du treillis de classes. Dans la section 4, nous concluons en présentant quelques résultats préliminaires obtenus sur les données C/Net.

2. Langages d'instances et de classes

On peut caractériser une classe en lui donnant une *définition extensionnelle* ou une *définition intensionnelle*. Dans le premier cas, on décrit une classe en donnant l'ensemble de ses instances. Dans le second cas, la classe est définie par l'ensemble de ses propriétés décrit en utilisant le langage des classes.

Un langage de classes $\mathcal{L}$ se définit par une syntaxe établissant les règles de construction de termes du langage pouvant décrire les propriétés des classes, et par une relation d'ordre partiel appelée *relation de subsomption* et notée $\preceq$. Un langage d'instances $\mathcal{L}'$ se définit par une syntaxe établissant les règles de construction de termes du langage pouvant décrire les propriétés des instances, et par *une relation d'appartenance* (notée $isa_{\mathcal{L}',\mathcal{L}}$) entre termes du langage et d'un langage $\mathcal{L}$ de classes. Nous noterons $desc_{\mathcal{L}}(C)$ la description de la classe C dans le langage $\mathcal{L}$, et $desc_{\mathcal{L}'}(i)$ la description de l'instance i dans $\mathcal{L}'$. Nous donnons des définitions générales qui fondent notre approche, et qui seront instanciées lors de la présentation des langages d'instances et de classes que nous avons choisi de mettre en oeuvre.

Nous définissons la relation de *subsomption* entre deux classes par une opération de conjonction (notée $\otimes$) entre les deux descriptions des classes qui renvoie la description de la classe la plus générale subsumée par les deux classes. Le calcul de subsomption entre deux classes est réalisé en appliquant l'opération de conjonction entre les deux descriptions des classes et en vérifiant que le résultat est équivalent (modulo une normalisation) à la classe subsumée.

Définition 1 (Subsomption dans un langage de classes $\mathcal{L}$) *Soient $C1$ et $C2$ deux classes, $C_1 \preceq_{\mathcal{L}} C_2$ (i.e. $desc_{\mathcal{L}}(C_1) \preceq desc_{\mathcal{L}}(C_2)$), $C2$ subsume $C1$, ssi $desc_{\mathcal{L}}(C2) \otimes desc_{\mathcal{L}}(C1)$ $\equiv desc_{\mathcal{L}}(C1)$*

Pour définir et calculer l'appartenance d'une instance à une classe, nous procédons par *abstraction-subsomption*. Plus précisément, pour savoir si une instance i appartient à une classe C, nous calculons l'abstraction de i dans $\mathcal{L}$ et nous testons si l'abstraction obtenue est subsumée par la description de C dans $\mathcal{L}$.

Définition 2 (Abstraction d'une instance dans un langage de classes $\mathcal{L}$) *L'abstraction d'une instance i dans un langage de classes $\mathcal{L}$ est une classe dont la description notée $abst_{\mathcal{L}}(i)$ est la description la plus spécifique de i dans $\mathcal{L}$.*

Définition 3 (Appartenance d'une instance à une classe) *Soit i une instance décrite dans un langage d'instances $\mathcal{L}'$ et C une classe décrite dans un langage de classes $\mathcal{L}$. $i\ isa_{\mathcal{L}',\mathcal{L}}\ C$ si et seulement si $abst_{\mathcal{L}}(i) \preceq desc_{\mathcal{L}}(C)$*

Définition 4 (Extension d'une classe) *Soit I un ensemble d'instances décrites dans $\mathcal{L}'$. On appelle extension d'une classe C décrite dans un langage de classe $\mathcal{L}$ l'ensemble des instances de I vérifiant l'appartenance à la classe C.*
$ext_I(C) = \{i \in I \mid i\ isa_{\mathcal{L}',\mathcal{L}}\ C\}$

Dans différentes étapes de notre approche, nous avons besoin de calculer la description d'une classe C dans un langage $\mathcal{L}$ à partir de son extension. Ce calcul est réalisé en appliquant l'opération de *généralisation maximalement spécifique* (*gms*) définie ci-dessous sur les abstractions des instances de C dans $\mathcal{L}$. Pour qu'une telle approche soit applicable, il faut que la gms existe toujours, soit unique et de taille finie. Les langages $\mathcal{L}_2$ et $\mathcal{L}_3$ que nous considérons vérifient cette condition.

Définition 5 (Généralisation maximalement spécifique) *Soient $T1$ et $T2 \in \mathcal{L}$, $gms(T1, T2)$ = $T3 \in \mathcal{L}$ ssi : $T1 \preceq T3$ et $T2 \preceq T3$ ($T3$ subsume à la fois $T1$ et $T2$) et $\nexists\, T4$, $T4 \in$*

$\mathcal{L}$ tel que $T1 \preceq T4$, $T2 \preceq T4$ et $T4 \prec T3$ (i.e. il n'existe pas de subsumant commun à $T1$ et $T2$ qui soit subsumé strictement par $T3$).

Calcul d'une description de classe à partir de son extension
Soit $ext(C) = \{i_1, \ldots, i_n\}$ l'extension d'une classe C, $desc_{\mathcal{L}}(C) \leftarrow gms(abst_{\mathcal{L}}(i_1), \ldots, abst_{\mathcal{L}}(i_n))$[3].

Nous décrivons maintenant les langages utilisés dans notre approche. La section 2.1 décrit la syntaxe du langage d'instances $\mathcal{L}_1$. Les sections 2.2 et 2.3 présentent les deux langages de classes $\mathcal{L}_2$ et $\mathcal{L}_3$ et leur relation avec $\mathcal{L}_1$ en définissant pour chacun l'opération $\otimes$ (relation de subsomption), le calcul d'abstraction des instances (relation d'appartenance) et l'opération de gms (calcul d'une description de classe à partir de son extension).

2.1. *Langage des instances $\mathcal{L}_1$*

SYNTAXE DE $\mathcal{L}_1$: Soit $\mathcal{A}$ un ensemble d'attributs. Soit pour chaque attribut att_j un domaine D_j. Un terme de $\mathcal{L}_1$ est de la forme : $\{att_1 = V_1, \ldots, att_n = V_n\}$ avec $\{att_1, \ldots, att_n\} \subseteq \mathcal{A}$ et pour tout j de 1 à n, $V_j \subseteq D_j$.
On définit une instance i dans $\mathcal{L}_1$ en lui associant un terme de $\mathcal{L}_1$.
Exemple : $desc_{\mathcal{L}_1}(p1) = \{class\text{-}base = \{pc\}, periph = \{souris, clavier\}\}$

2.2. *Langage de classes $\mathcal{L}_2$*

SYNTAXE DE $\mathcal{L}_2$: Soit $\mathcal{A}$ un ensemble d'attributs. Un terme de $\mathcal{L}_2$ est de la forme : $\{att_1, \ldots, att_n\}$ avec $\{att_1, \ldots, att_n\} \subseteq \mathcal{A}$. On définit une classe C dans $\mathcal{L}_2$ en lui associant un terme de $\mathcal{L}_2$.

RELATION DE SUBSOMPTION DANS $\mathcal{L}_2$: Cf définition 1 avec $\otimes = \cup$ et $\mathcal{L} = \mathcal{L}_2$.

RELATION D'APPARTENANCE : Cf définition 3 avec le calcul d'abstraction suivant : Soit $desc_{\mathcal{L}_1}(i) = \{att_1 = V_1, \ldots, att_n = V_n\}$, $abst_{\mathcal{L}_2}(i) = \{att_1, \ldots, att_n\}$

GÉNÉRALISATION MAXIMALEMENT SPÉCIFIQUE : soient $T1$ et $T2$ deux termes de $\mathcal{L}_2$, $gms(T1, T2) = T1 \cap T2$.

2.3. *Langage des classes $\mathcal{L}_3$*

SYNTAXE DE $\mathcal{L}_3$: Soit $\mathcal{A}$ un ensemble d'attributs. Soit pour chaque attribut att_j un domaine D_j. Un terme de $\mathcal{L}_3$ est de la forme : $\{att_1^{[suff]} = V_1, \ldots, att_n^{[suff]} = V_n\}$

3. L'opération de gms étant commutative et associative, on peut appliquer cette opération binaire de manière récursive : $gms(T1, \ldots, Tn) = gms(T1, gms(T2, \ldots, Tn))$

avec $\{att_1, \ldots, att_n\} \subseteq \mathcal{A}$, pour tout j de 1 à n, $V_j \subseteq D_j$ et $suff \in \{*,+,?\}$.
Soit une classe C, on définit C dans $\mathcal{L}_3$ en lui associant un terme de $\mathcal{L}_3$.
Les contraintes liées aux propriétés d'une classe peuvent porter sur l'absence d'un attribut, sur les valeurs possibles qu'il peut prendre ou sur sa cardinalité.
$att_i^* = V_i$: les instances de la classe peuvent avoir ou non cet attribut et l'attribut peut être mono ou multi valué. Si l'attribut est présent, toutes les valeurs doivent appartenir à l'ensemble V_i
$att_i^+ = V_i$: les instances de la classe ont au moins une valeur pour cet attribut et toutes les valeurs doivent appartenir à V_i
$att_i^? = V_i$: les instances de la classe peuvent avoir ou non cet attribut, celui-ci est mono valué et sa valeur doit appartenir à V_i
$att_i = V_i$: les instances de la classe ont une valeur et une seule pour cet attribut et cette valeur doit appartenir à V_i
L'attribut n'est pas présent : aucune instance de la classe ne possède cet attribut.

RELATION DE SUBSOMPTION : Pour définir la relation de subsomption dans $\mathcal{L}_3$, on se ramène à la définition 1. L'opération $\otimes$ de conjonction de termes décrite ci-dessous prend en compte l'ordre de généralité des différents suffixes suivant : $att_i^+ \prec att_i^*$, $att_i^? \prec att_i^*$, $att_i \prec att_i^+$, $att_i \prec att_i^?$, $non(att_i)$[4] $\prec att_i^?$. La fonction conj-suff(suffi,suffj) renvoie le suffixe le plus général subsumé par suffi et suffj. La fonction normalisation(nouv-att) renvoie l'attribut sous une forme normalisée. La normalisation consiste à décrire de manière unique des propriétés exprimant la même contrainte et pouvant s'exprimer de manières syntaxiquement différentes (exemple : $couleur^+ = \{bleu\}$ est équivalente à $couleur = \{bleu\}$). Nous choisissons de garder la description la plus spécifique. La normalisation permet également de détecter des incohérences (exemple $couleur^+ = \emptyset$ est équivalente à *incohérent*). Si la normalisation d'un des attributs d'une description conduit à la valeur *incohérent*, celle-ci est normalisée par la constante *bottom* qui correspond aux classes dont l'extension est vide.

Calcul de la conjonction de termes dans $\mathcal{L}_3$: $T3 = T1 \otimes T2$
Si $T1$ = *bottom* ou $T2$ = *bottom* **alors** retourner *bottom* **sinon**

début
$T3 \leftarrow \emptyset$
{* Gestion des attributs présents dans un seul des deux termes *}
Pour tout $att_j^{[suffj]} = V_j \in T1$ tel que att_j n'est pas dans $T2$ **faire**
  si suffj = $\emptyset$ ou suffj = + **alors** *retourner bottom*
Pour tout $att_j^{[suffj]} = V_j \in T2$ tel que att_j n'est pas dans $T1$ **faire**
  si suffj = $\emptyset$ ou suffj = + **alors** *retourner bottom*
{* Gestion des attributs communs *}
Pour tout $att_j^{[suffj1]} = V_{j1} \in T1$ tel que $att_j^{[suffj2]} = V_{j2} \in T2$ **faire**
  suffj3 $\leftarrow$ conj-suff(suffj1,suffj2)

4. Le non n'est pas un connecteur du langage mais une notation pour indiquer l'absence de l'attribut att_i.

$V_{j3} \leftarrow V_{j1} \cap V_{j2}$
nouv-att $\leftarrow att_j^{[suffj3]} = V_{j3}$
normalisation(nouv-att)
si nouv-att = incohérent **alors** *retourner bottom*
sinon $T3 \leftarrow T3 \cup$ nouv-att

fin

Exemple : $desc_{\mathcal{L}_3}(C1)$ = $\{couleur^{+} = \{bleu, rose\}, taille = \{10\}\}$, $desc_{\mathcal{L}_3}(C2)$ = $\{couleur^{*} = \{bleu, rouge, rose\}, taille^{?}$ = {10,12}}. $C2$ subsume $C1$ puisque $desc_{\mathcal{L}_3}(C2) \otimes desc_{\mathcal{L}_3}(C1) \equiv \{couleur^{+} = \{bleu, rose\}, taille$ = {10}}

RELATION D'APPARTENANCE : Cf définition 3 avec le calcul d'abstraction suivant : soit $desc_{\mathcal{L}_1}(i)$ = $\{att_1 = V_1, \ldots, att_n = V_n\}$
$abst_{\mathcal{L}_3}(i)$ = $\{att_1^{[suff1]} = V_1, \ldots, att_n^{[suffn]} = V_n\}$ avec $\forall j \in 1 \ldots n$, $suff_j$ = + si $|V_j| \geq 2$
GÉNÉRALISATION MAXIMALEMENT SPÉCIFIQUE : La fonction gms-suff(suff1,suff2) renvoie le suffixe le plus spécifique qui subsume suff1 et suff2. Les termes $T1$ et $T2$ sont normalisés.

Calcul de gms(T1,T2) :
Si $T1$ (resp. $T2$) = *bottom* **alors** retourner $T2$ (resp. $T1$) **sinon**
début
$gms \leftarrow \emptyset$
{* Attributs dans $T1$ et pas dans $T2$ *}
Pour tout $att_j^{[suffj1]} = V_{j1} \in T1$ tel que att_j n'est pas dans $T2$ **faire**
si suffj1=$\emptyset$ ou suffj1=? **alors** $gms \leftarrow gms \cup att_j^{?} = V_{j1}$
sinon $gms \leftarrow gms \cup att_j^{*} = V_{j1}$
{* Attributs dans $T2$ et pas dans $T1$ *}
Pour tout $att_j^{[suffj2]} = V_{j2} \in T2$ tel que att_j n'est pas dans $T1$ **faire**
si suffj2=$\emptyset$ ou suffj2=? **alors** $gms \leftarrow gms \cup att_j^{?} = V_{j2}$
sinon $gms \leftarrow gms \cup att_j^{*} = V_{j2}$
{* Attributs communs à $T1$ et $T2$ *}
Pour tout $att_j^{[suffj1]} = V_{j1} \in T1$ tel que $att_j^{[suffj2]} = V_{j2} \in T2$ **faire**
suffj3 $\leftarrow$ gms-suff(suffj1,suffj2)
$V_{j3} \leftarrow V_{j1} \cup V_{j2}$
$gms \leftarrow gms \cup att_j^{suff3} = V_{j3}$
fin

3. Construction de hiérarchies

A partir d'instances étiquetées par une classe de base, on veut calculer la description la plus spécifique dans $\mathcal{L}_2$ de chaque classe de base, trouver des regroupements non disjoints de ces classes de base, calculer la description la plus spécifique dans $\mathcal{L}_2$ de chaque classe obtenue par regroupement, organiser les classes obtenues dans une hiérarchie dite grossière, transformer automatiquement la hiérarchie obtenue précédemment ou une partie de cette hiérarchie suivant les besoins de l'utilisateur en une hiérarchie dans $\mathcal{L}_3$ (on a les mêmes noeuds et arcs mais les noeuds sont décrits dans $\mathcal{L}_3$), affiner la hiérarchie transformée (donc ajouter des noeuds intermédiaires). Les quatre premières tâches sont décrites dans la section 3.1, les deux dernières dans la section 3.2.

3.1. *Construction d'une hiérarchie grossière dans $\mathcal{L}_2$*

• **Calcul de la description des classes de base dans $\mathcal{L}_2$**
Les instances étant déjà étiquetées par des classes de base, nous exploitons cette première agrégation. Nous ne disposons que de la définition extensionnelle de ces classes, extension que l'on notera $ext_{concepteur}(C)$. Il nous faut donc tout d'abord calculer leur description intensionnelle en utilisant l'opération de calcul de la description la plus spécifique dans $\mathcal{L}_2$ telle qu'elle a été définie en section 2. Or, si l'on cherche à calculer l'extension de la classe de base $ext_I(C)$ à partir de la description calculée précédemment, nous ne serons en aucun cas assurés que $ext_{concepteur}(C) = ext_I(C)$. En effet d'autres instances de I peuvent appartenir à la classe ainsi définie[5]. Pour conserver l'adéquation entre l'intension et l'extension, nous transformons l'attribut $class\text{-}base$ par sa valeur dans l'abstraction des instances de cette classe. Exemple : $desc_{\mathcal{L}_1}(p1) = \{class\text{-}base = \{portable\}, periph\text{-}E = \{souris, clavier\}, syst\text{-}exp = \{WindowsNT\}\}$, $abst_{\mathcal{L}_2}(p1) = \{portable, periph\text{-}E, syst\text{-}exp\}$. Nous obtenons alors $ext_{concepteur}(C) = ext_I(C)$.

• **Construction de la hiérarchie à partir des descriptions des classes de base**
Les treillis de concepts [BAR 70] ou treillis de Galois [WIL 92] sont des structures mathématiques qui ont été largement utilisées en apprentissage de concepts. Ils permettent de représenter l'ensemble des classes non-disjointes sous jacentes à un ensemble de données sachant que l'on peut définir par la suite des procédures de sélection permettant de réduire le nombre de classes du treillis. Pour définir un treillis de Galois, nous devons tout d'abord définir son contexte. Dans notre approche, le contexte Co est défini par le triplet $(CBase, \mathcal{A}, \mathcal{R})$ où $CBase$ est l'ensemble des classes de base, $\mathcal{A}$ l'ensemble des attributs et $\mathcal{R}$ est une relation de $CBase \times \mathcal{A}$ qui indique pour chaque couple $\{C, att\}$ si l'attribut att de $\mathcal{A}$ appartient à $desc_{\mathcal{L}_2}(C)$
Soit f l'application qui à tout élément C de $CBase$ associe $f(C) = desc_{\mathcal{L}_2}(C)$ et g l'application qui à tout attribut att de $\mathcal{A}$ associe $g(att)$ ={ C $\in CBase$ / $att \in desc_{\mathcal{L}_2}(C)\}$. Le couple (f, g) est une correspondance de Galois et leur définition peut être étendu aux parties de $\mathcal{A}$ et de $CBase$. Soit $A1$ une partie de $\mathcal{A}$ et $C1$ une partie de $CBase$. Le couple $\{A1, C1\}$ est un *concept* ssi $f(C1) = A1$ et $g(A1) = C1$. L'ensemble de tous les concepts $\mathcal{C}$ muni de la relation de subsomption dans $\mathcal{L}_2$ est un treillis de Galois. Ce treillis de Galois comporte un nombre maximum de classes

5. Ainsi, dans C/net, la description la plus spécifique de la classe de base *Cable d'entrée* est identique à celle de *Cable de sortie*. On retrouverait donc par calcul que l'extension de Cable d'entree comporte toutes les instances de Cable de sortie.

de $2^{min(m,n)}$ avec $m = |\mathcal{A}|$ et $n = |CBase|$. Pour ne pas exprimer le treillis complet, on peut introduire des opérations de sélection concernant soit les attributs, soit les concepts (i.e. les classes). Pour l'instant, nous avons simplement utilisé une sélection quantitative sur les classes : une classe $\{A_i, C_i\}$ apparaîtra dans la hiérarchie ssi $|C_i| > n$, c'est à dire si elle subsume plus de n classes de base. Il a éte montré que l'utilisation d'un tel seuil réduisait de manière notable la taille du treillis [WIL 82]. Pour aboutir à ce résultat, nous utilisons l'algorithme du Frequent Item Set dans sa version bottom-up [AGR 94]. Cet algorithme va rechercher les ensembles maximaux d'attributs qui apparaissent ensemble dans plus de n classes. Les classes obtenues par cette agrégation forment bien un sous-ensemble des nœuds du treillis de Galois de contexte Co satisfiant le critère de sélection. Pour les avoir toutes, nous conservons toutes les classes intermédiaires calculées par le Frequent Item Set (les ensembles d'attributs non-maximaux) qui satisfont la définition d'un concept dans un treillis de Galois. Le calcul de subsomption dans $\mathcal{L}_2$ nous permet d'organiser ces classes dans une hiérarchie.

3.2. *Affinage de la hiérarchie dans* $\mathcal{L}_3$

Le but de cette seconde étape est d'affiner la hiérarchie grossière obtenue à l'étape précédente en utilisant un langage de classe plus expressif. Cette étape est réalisée en fonction des besoins de l'utilisateur qui détermine la partie de la hiérarchie qui l'intéresse. Pour cela, il choisit un nœud père et un ou plusieurs nœuds fils plus spécifiques. Dans un premier temps, nous transformons cette partie de la hiérarchie en une hiérarchie similaire (les nœuds sont les mêmes) décrite cette fois dans $\mathcal{L}_3$. Cette étape est décrite dans la section 3.2.1. Dans un second temps, nous affinons cette hiérarchie en calculant des nœuds intermédiaires situés entre le nœud père et les nœuds fils sélectionnés (section 3.2.2). Auparavant nous donnons quelques définitions préliminaires. La définition 1 (section 2) présente la subsomption d'un point de vue intensionnel. Dans ce cadre, une description $D2$ est équivalente à une description $D1$ ssi $\forall I$, $ext_I(D1) = ext_I(D2)$. Dans le cadre d'une application, nous raisonnons dans un domaine I particulier correspondant aux instances dont on dispose. Dans ce domaine I, des descriptions qui ne sont pas équivalentes peuvent le devenir. On parle alors de *classe d'équivalence empirique*. La notion de classe d'équivalence empirique a été introduite dans [CHA 95] en vue de réduire l'espace de recherche des hypothèses en apprentissage de concepts. Dans notre cadre, sa définition est la suivante :

Définition 6 (Classe d'équivalence empirique dans un langage $\mathcal{L}$) *Soit I un ensemble d'instances décrites dans $\mathcal{L}_1$. Une classe d'équivalence empirique est formé de l'ensemble des descriptions de $\mathcal{L}$ ayant la même extension dans I*

Parmi toutes les descriptions d'une classe d'équivalence empirique, il existe dans certains langages une et une seule description plus spécifique que toutes les autres descriptions de la classe d'équivalence. Dans ce cas, on peut choisir cette description comme *représentant* de la classe. Nous rappelons que nous travaillons sur des descriptions normalisées et que toutes les opérations auxquelles on fait référence renvoient

des descriptions normalisées, ce qui est une condition nécessaire pour considérer que le représentant obtenu est unique.
L'opération de *fermeture* d'une description quelconque de $\mathcal{L}$ consiste à remplacer cette description par le représentant de sa classe d'équivalence empirique dans $\mathcal{L}$. Cette opération est réalisée en deux étapes. Dans un premier temps, nous calculons l'extension de la description puis nous calculons le gms des abstractions des instances de l'extension. La description obtenue est appelée *description fermée*. Toute description fermée Df est telle que : $\forall D$ telle que $ext_I(D) = ext_I(Df)$, $Df \prec D$

3.2.1. *Description dans $\mathcal{L}_3$ de la hiérarchie sélectionnée*

Le but de cette étape est de calculer la description la plus spécifique dans $\mathcal{L}_3$ de tous les nœuds choisis par l'utilisateur. Pour cela, nous calculons pour chaque nœud N son extension à partir de sa description dans $\mathcal{L}_2$ en utilisant les opérations décrites dans la section 2.2. Notons que l'extension peut être un sur-ensemble de l'union des classes agrégées. A partir de chacune de ces extensions nous calculons la description la plus spécifique dans $\mathcal{L}_3$ de la classe en utilisant l'opération de gms définie dans la section 2.3. Notons que l'opération décrite ici est une extension de l'opération de fermeture qui permet de passer d'une description dans un langage à une description dans ce même langage.

3.2.2. *Ajout de nœuds intermédiaires*

Pour construire des nœuds intermédiaires dans la hiérarchie obtenue, nous calculons des spécialisations du nœud père telles que ces spécialisations subsument au moins un des nœuds fils. Nous distinguons la spécialisation intensionnelle dans $\mathcal{L}_3$ qui s'appuie sur la relation de subsomption classique et la spécialisation empirique définie ci-dessous.

Définition 7 (spécialisation empirique dans $\mathcal{L}_3$) *Soit I un ensemble d'instances. Soient $C1$ et $C2$ deux descriptions de $\mathcal{L}_3$, $C2$ est une spécialisation empirique de $C1$ si et seulement si : $ext_I(C2) \subset ext_I(C1)$*

Pour spécialiser une description, nous pouvons pour chaque attribut de la description remplacer son suffixe par un suffixe plus spécifique ce qui peut conduire à enlever cet attribut, enlever une ou plusieurs valeurs de son ensemble des valeurs possibles. Notons que l'absence d'un attribut dans une description est une contrainte très spécifique et que, contrairement à un cadre classique, l'ajout d'un attribut lorsque celui est absent n'est pas une spécialisation.
L'application d'une ou de plusieurs de ces opérations sur une description non fermée ne conduit pas forcément à une spécialisation empirique (i.e. on peut aboutir à un terme de la même classe d'équivalence empirique).
Par contre, l'application d'une opération de spécialisation sur une description fermée conduit toujours à une spécialisation empirique.
Preuve : soit $D1$ une description fermée, soit $ext_I(D1)$ son extension. Supposons qu'il existe une description $D2$ obtenue par application d'un ou de plusieurs des opérateurs de spécialisation ($D2 \prec D1$) telle que $ext_I(D1) = ext_I(D2)$. $D1$ est fermée

d'oú $\forall D$ tel que $ext_I(D) = ext_I(D1)$, $D1 \prec D$. On a donc $D1 \prec D2$ ce qui est contradictoire avec notre hypothèse $D2 \prec D1$.
Le processus d'ajout de nœuds intermédiaires consiste alors, en partant du nœud père fermé, à alterner à chaque étape une phase de spécialisation et une phase de *fermeture* de la description obtenue. De plus, nous ne conservons dans le treillis que les descriptions qui vérifient la contrainte de subsumer au moins un des nœuds fils. La hiérarchie obtenue est donc un sous-ensemble d'une hiérarchie comportant toutes les spécialisations. D'autres travaux ont défini et utilisé des espaces réduits notamment dans le cadre de l'apprentissage [BRE 98, GAN 93]. Dans [BRE 98], un opérateur de spécialisation qui travaille sur un espace fermé est présenté et implémenté dans un algorithme d'apprentissage de concepts dans le cadre d'un langage atomique. La définition de cet opérateur dans $\mathcal{L}_3$ est une extension de ces travaux. Dans [GAN 93], l'espace réduit calculé est utilisé pour construire un ensemble d'implications logiques vraies sur les instances.
Notons que dans notre approche, en raison de l'expressivité de $\mathcal{L}_3$, les contraintes de fermeture et de subsomption d'au moins un des nœuds fils ne suffisent pas pour restreindre suffisamment le nombre de nœuds obtenus si l'on applique systématiquement toutes les spécialisations possibles à chaque étape. Nous présentons dans la section suivante quelques critères pouvant permettre de réduire le nombre de nœuds du treillis résultat.

4. Conclusion et perspectives

Le but géneral de notre approche consiste à organiser des données sous la forme d'une hiérarchie grossière pouvant être affinée à la demande d'un utilisateur. Le probléme de la réduction de la taille d'un treillis de concepts a conduit à des techniques permettant de ne sélectionner qu'un sous-ensemble d'attributs ou de classes sur des critères quantitatifs ou qualitatifs [BRE 98, NGU 94, NJA 00]. Ici, l'objectif n'est pas uniquement de réduire le nombre de classes de la première hiérarchie, mais d'effectuer un changement de niveau de granularité. Dans nos travaux, nous modifions la granularité principalement en effectuant un changement de langage (passage de $\mathcal{L}_2$ à $\mathcal{L}_3$). Le choix des nœuds aussi bien dans la hiérarchie grossière que dans la hiérarchie affinée est basé sur la notion de classe d'équivalence empirique. Cependant, dans $\mathcal{L}_3$, les classes d'équivalences sont plus fines (des instances appartenant à la même classe d'équivalence dans $\mathcal{L}_2$ peuvent appartenir à des classes d'équivalence différentes dans $\mathcal{L}_3$). Le nombre de nœuds augmente alors en même temps que la précision de leur description.
Nous avons réalisé une implémentation C^{++} de toutes les opérations définies sur les langages $\mathcal{L}_1$, $\mathcal{L}_2$ et $\mathcal{L}_3$. Nous avons appliqué la première étape de construction d'une hiérarchie grossière dans $\mathcal{L}_2$ aux 2274 produits C/Net en utilisant un seuil de 10% pour le critère de sélection. La hiérarchie obtenue comporte 46 classes avec une hauteur maximum de 5. Les différentes opérations de base définies pour la construction d'une hiérarchie affinée ont également été implémentées. Nous nous sommes intéressés à l'impact de l'opération de fermeture sur le nombre de classes élaguées. En

sélectionnant au hasard plusieurs classes de la hiérarchie obtenue lors de premiére étape, le nombre de classes spécialisées élaguées est à chaque fois proche de 50% ce qui représente un gain intéressant.

Il s'agit maintenant d'allier à notre changement de langage des méthodes pertinentes de réduction de taille du treillis qui nous permettrons, pour la premiére étape, d'améliorer la pertinence des classes visualisées et pour la deuxième étape d'exploiter les opérations définies dans la hiérarchie affinée. Pour cette deuxième étape, nous pouvons, par exemple, réduire le nombre de spécialisations en réduisant le nombre de valeurs symboliques ou numériques à prendre en compte (treillis de type, intervalles [POL 98]).

Remerciements

Nous tenons à remercier Henry Soldano (LIPN Paris XIII) pour les nombreuses discussions que nous avons eues ensemble et qui ont fortement contribué à l'avancement de ce travail

5. Bibliographie

[AGR 94] AGRAWAL R., SRIKANT R., « Fast Algorithms for Mining Association Rules », *20th VLDB Conference*, Santiago, Chile, 1994.

[BAR 70] BARBUT M., MONJARDET B., *Ordre et Classification*, Paris, Hachette Université, 1970.

[BRE 98] BREZELLEC P., SOLDANO H., « Tabata : a learning algorithm performing a bidirectional search in a reduced search space using a tabu stratégie », *ECAI'98*, Brighton, Angleterre, 1998, p. 420–424.

[CHA 95] CHAMPESME M., BRÉZELLEC P., SOLDANO H., « Réduction de l'espace de recherche en apprentissage : résultats théoriques et expérimentaux », *Actes Journées Acquisition Validation Apprentissage, JAVA95*, Grenoble, France, 1995, p. 192–196.

[GAN 93] GANASCIA J., « TDIS: an Algebraic Formalization », *International Joint Conference on Artificial Intelligence*, Vancouver, BC, Canada, 1993, p. volume 2, 1008–1013.

[NGU 94] NGUIFO E. M., « Galois Lattice: A framework for Concept Learning. Design, Evaluation and Refinement », *Proceedings of ICTAI'94*, New Orleans, Louisiana, LA, 1994, p. 461–467.

[NJA 00] NJAMEN G. N., *Contribution à l'apprentissage symbolique automatique par l'usage du treillis de Galois*, Thèse, Université d'Artois, France, 2000.

[POL 98] POLAILLON G., *Organisation et interprétation par les treillis de Galois de données de type multivalué, intervalle ou histogramme*, Thèse, Université Paris 9, France, 1998.

[WIL 82] WILLE R., « Restructuring Lattice Theory », *Symposium on Ordered Sets*, University of Calgary, Boston, 1982, p. 445–470.

[WIL 92] WILLE R., « Concept Lattices and Conceptual Knowledge Systems », *Computer Mathematic Applied*, vol. 23, 1992.