Automatic Construction and Refinement of a Class Hierarchy over Semistructured Data

Nathalie Pernelle, Marie-Christine Rousset, Veronique Ventos
L.R.I., C.N.R.S. & University of Paris-Sud
Building 490, 91405, Orsay Cedex, France
Email: {pernelle, mcr,ventos}@lri.fr

1 Introduction

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 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. Our approach has been implemented and experimented on real data in the setting of the GAEL project [1] which aims at building flexible electronic catalogs organized as a hierarchy of classes of products. Our experiments have been conducted on real data coming from the C/Net (http://www.cnet.com) electronic catalog of computer products.

2 Languages of instances and classes

In this section, we define the language of instances, $\mathcal{L}_1$, in which we describe semistructured data, and the two languages of classes $\mathcal{L}_2$ and $\mathcal{L}_3$ that we use to describe classes over those data, at two levels of abstraction. First, we provide some notations and preliminaries.

2.1 Preliminaries and notations

Given a language of instances, a language of classes $\mathcal{L}$ defines the expressions that are allowed as *class descriptions*. A class description is intended to represent in an abstract and concise way the properties that are common to the set of its instances. A *membership relation*, denoted by $isa_{\mathcal{L}}$, establishes the necessary connection between a given language of instances and an associated language of classes $\mathcal{L}$.

Definition 1 (Extension of a class description) *Let I be a set of instances, and C a $\mathcal{L}$ class description. The extension of C w.r.t I is the following set:*

$$ext_I(C) = \{i \in I \mid i \, isa_{\mathcal{L}} \, C\}$$

[1]GAEL is funded by the French Ministry of Industry ; it is a joint project with the Verso database group of INRIA and a startup, MatchVision, specialized in Electronic Commerce

The subsumption relation is a preorder relation between class descriptions, induced by the inclusion relation between class extensions.

Definition 2 (Subsumption between classes) *Let C_1 and C_2 be two $\mathcal{L}$ class descriptions. C_1 is subsumed by C_2, denoted $C_1 \preceq_{\mathcal{L}} C_2$, iff for every set I of instances, $ext_I(C_1) \subseteq ext_I(C_2)$.*

In sections 2.3, and 2.4, we will provide a constructive characterization of subsumption for the two languages of classes that we consider.

The notion of *abstraction* of an instance in a language of classes $\mathcal{L}$ corresponds, when it exists, to the most specific class description in $\mathcal{L}$ which it is an instance of.

Definition 3 (Abstraction of an instance) *Let i be an instance, the $\mathcal{L}$ class description C is an* abstraction *of i in $\mathcal{L}$ (for short $C = abs_{\mathcal{L}}(i)$) iff*

1. *$i \; isa_{\mathcal{L}} \; C$, and*
2. *if D is a class description such that $i \; isa_{\mathcal{L}} \; D$, then $C \preceq_{\mathcal{L}} D$.*

The notion of *least common subsumer* (a.k.a msg) will be the basis for gathering classes in our clustering algorithm.

Definition 4 (Least Common Subsumer) *Let $C_1, \ldots, C_n$ be class descriptions in $\mathcal{L}$. The $\mathcal{L}$ class description C is a* least common subsumer *of $C_1, \ldots, C_n$ in $\mathcal{L}$ (for short $C = lcs_{\mathcal{L}}(C_1, \ldots, C_n)$) iff*

1. *$C_i \preceq_{\mathcal{L}} C$ for all $1 \leq i \leq n$, and*
2. *if D is a class description satisfying $C_i \preceq_{\mathcal{L}} D$ for all $1 \leq i \leq n$, then $C \preceq_{\mathcal{L}} D$*

2.2 The language of instances

The data that serve as instances of the classes that we build are semistructured data in the following sense: each data is described by a set of pairs (Attribute, Values) but the attributes used for describing the data may vary from an item to another. In addition, each data is typed (i.e., labelled by the name of a basic type).

Definition 5 (Terms of $\mathcal{L}_1$) *Let $\mathcal{B}$ be a finite set of basic types, $\mathcal{A}$ a finite set of* attributes, *and $\mathcal{V}$ a set of* values. *A term of $\mathcal{L}_1$ is of the form:*

$$\{c, att_1 = V_1, \ldots, att_n = V_n\}$$

where $c \in \mathcal{B}$, $\forall i \in [1..n]$, $att_i \in \mathcal{A}$ and $V_i \subseteq \mathcal{V}$.

The description of an instance is a term of $\mathcal{L}_1$. For example, we can find a product in the C/Net catalog, whose $\mathcal{L}_1$ description is:

$\{RemovableDiskDrive,$
$CD/DVD/Type{=}\{CDRW\},$
$StorageRemovableType{=}\{SuperDisk\},$
$Compatibility{=}\{MAC, PC\}\}$

In the following, we will consider that the type c of a $\mathcal{L}_1$ description is a boolean attribute.

2.3 The language of classes $\mathcal{L}_2$

Definition 6 (Class description in $\mathcal{L}_2$) *A $\mathcal{L}_2$ class description (of size n) is a tuple of attributes $\{att_1, \ldots, att_n\}$, where $\forall i \in [1..n]$, $att_i \in \mathcal{A}$.*

The connection between the language of instances $\mathcal{L}_1$ and the language of classes $\mathcal{L}_2$ is based on the following definition of the membership relation.

Definition 7 (Membership relation for $\mathcal{L}_2$) *Let i be an instance description in $\mathcal{L}_1$. Let C be a $\mathcal{L}_2$ class description: i is an instance of C iff every attribute appearing in C also appears in i.*

The following proposition, whose proof is straightforward, characterizes subsumption, least common subsumer and abstraction in $\mathcal{L}_2$.

Proposition 1 (Properties of $\mathcal{L}_2$) .
- *Let C_1 and C_2 be two $\mathcal{L}_2$ class descriptions. $C_1 \preceq_{\mathcal{L}_2} C_2$ iff every attribute of C_2 is also an attribute of C_1.*
- *Let $\{att_1 = V_1, \ldots, att_n = V_n\}$ be an instance description in $\mathcal{L}_1$. Its abstraction in $\mathcal{L}_2$ is unique: it is $\{att_1, , \ldots, att_n\}$.*
- *Let $C_1, \ldots, C_n$ be n $\mathcal{L}_2$ class descriptions. Their least common subsumer is unique: it is made of the set of attributes that are common to all the C_i's.*

2.4 The language of classes $\mathcal{L}_3$

$\mathcal{L}_3$ is richer than $\mathcal{L}_2$ on different aspects: it makes possible to restrict the possible values of an attribute ; it enables to distinguish the number of values of an attribute through different suffixes $(*, +, ?, \epsilon)$ whose notation is inspired by the one used in XML for describing document type definitions (DTDs), and whose formal semantics corresponds to standard description logics constructors. In fact, as it will become clearer in the following, $\mathcal{L}_3$ is a subset of the C-CLASSIC description logic [7].

Definition 8 (Class description in $\mathcal{L}_3$) *A $\mathcal{L}_3$ class description (of size n) is a tuple*

$$\{att_1^{suff_1} : V_1, \ldots, att_n^{suff_n} : Vn\}$$

where $\forall i \in [1..n]$, $att_i \in \mathcal{A}$, $V_i \subseteq \mathcal{V}$, and $suff_i \in \{*, +, ?, \epsilon\}$

The following definition formalizes the membership relation between an instance and a class description in $\mathcal{L}_3$.

Definition 9 (Membership relation for $\mathcal{L}_3$) *Let i be an instance description in $\mathcal{L}_1$. Let C be a $\mathcal{L}_3$ class description. i is an instance of C iff every attribute in i appears in C and for every term $att^{suff} : V$ appearing in C,*
- when $suff = *$, if there exists V' s.t $att = V' \in i$, then $V' \subseteq V$,
- when $suff = +$, there exists $V' \subseteq V$ s.t $att = V' \in i$,
- when $suff = ?$, if there exists V' s.t $att = V' \in i$, then V' is a singleton and $V' \subseteq V$,
- when $suff = \epsilon$, there exists V' singleton s.t $V' \subseteq V$ and $att = V' \in i$.

The product described in 2.2 is an instance of the $\mathcal{L}_3$ class description C_1:

$\{ RemovableDiskDrive^{\epsilon}{:}\{true\},$
$CD/DVD/ReadSpeed^{?}{:}\{20x, 32x, 24x\},$
$CD/DVD/Type^{\epsilon}{:}\{CDROM, CDRW\},$
$Compatibility^{+}{:}\{MAC, PC\},$
$StorageRemovableType^{\epsilon}{:}\{SuperDisk, ZIP, JAZ\}\}$

It represents the set of products that have in their description *(i)* necessarily the monovalued and boolean attribute $RemovableDiskDrive$ whose value must be $true$, *(ii)* possibly the attribute $CD/DVD/ReadSpeed$, and if that is the case, this attribute is monovalued and its value belongs to the set $\{20x, 32x, 24x\}$, *(iii)* necessarily the attribute $CD/DVD/Type$, which is monovalued and takes its value in the set $\{CDROM, CDRW\}$, *(iv)* necessarily the attribute $Compatibility$, which can be multivalued and takes its value(s) in the set $\{MAC, PC\}$, *(v)* necessarily the attribute $StorageRemovableType$, which is monovalued and takes its value in the set $\{SuperDisk, ZIP, JAZ\}$.

The following propositions state the main properties of $\mathcal{L}_3$. Their proofs follow from results in tractable description logics where structural subsumption is complete.

Proposition 2 (Characterization of subsumption in $\mathcal{L}_3$) *Let C_1 and C_2 be two $\mathcal{L}_3$ class descriptions. $C_1 \preceq_{\mathcal{L}_3} C_2$ iff all the attributes appearing in C_1 appear also in C_2 and for every pair $att^{suff} : V$ appearing in C_2,*
- when $suff = *$, if there exists $att^{suff'} : V' \in C_1$, then $V' \subseteq V$,
- when $suff = +$, there exists $V' \subseteq V$ s.t $att^{+} : V' \in C_1$ or $att^{\epsilon} : V' \in C_1$
- when $suff = ?$, if there exists $att^{suff'} : V' \in C_1$, then $suff' = ?$ or $suff' = \epsilon$, and $V' \subseteq V$,
- when $suff = \epsilon$, there exists V' s.t $V' \subseteq V$ and $att^{\epsilon} : V' \in C_1$.

The complexity of checking subsumption in $\mathcal{L}_3$ is quadratic w.r.t the maximal size of class descriptions.

Proposition 3 (Characterization of abstraction in $\mathcal{L}_3$) *Let $\{att_1 = V_1, \ldots, att_n = V_n\}$ be an instance description in $\mathcal{L}_1$. Its abstraction in $\mathcal{L}_3$ is unique: $abs_{\mathcal{L}_3} = \{att_1^{suff_1} : V_1, \ldots, att_n^{suff_n} : V_n\}$, where $\forall i \in [1..n]$, if $\mid V_i \mid \geq 2$ then $suff_i = +$ else $suff_i = \epsilon$.*

Proposition 4 (Characterization of lcs in $\mathcal{L}_3$) *Let $C_1, \ldots, C_n$ be n $\mathcal{L}_3$ class descriptions. Let A be the set of attributes belonging to at least one description C_i. $C_1, \ldots, C_n$ have a unique least common subsumer in $\mathcal{L}_3$, whose description is characterized as follows:*

- *for every attribute $att \in A$, let V be the union of the sets of values associated with att in the class descriptions C_i's: $V = \bigcup_1^n \{v \in V_i \mid att^{suff} : V_i \in C_i\}$.*
  - $att^{\epsilon}{:}V \in lcs(C_1, \ldots, C_n)$ *iff* $att^{\epsilon}{:}V_i \in C_i\ \forall i \in [1..n]$.
  - $att^{?}{:}V \in lcs(C_1, \ldots, C_n)$ *iff* $(\forall i \in [1..n]\ att^{*}{:}V_i \notin C_i$ *and* $att^{+}{:}V_i \notin C_i)$,*and*
    - *either* $\exists i \in [1..n]$ *s.t.* $att^{?}{:}V_i \in C_i$,
    - *or* $\exists i \in [1..n]$ *s.t.* $att^{s}{:}V' \notin C_i$ *for any s.*
  - $att^{*}{:}V \in lcs(C_1, \ldots, C_n)$ *iff*
    - *either* $\exists i \in [1..n]$ *s.t.* $att^{*}{:}V_i \in C_i$,
    - *or* $\exists i \in [1..n]$ *s.t.* $att^{+}{:}V_i \in C_i$, *and* $\exists j \in [1..n]$ *s.t.* $att^{?}{:}V_j \in C_j$ *or* $att^{s'}{:}V' \notin C_j$ *for any suffix s'.*
  - $att^{+}{:}V \in lcs(C_1, \ldots, C_n)$ *iff* $\exists i \in [1..n]$ *s.t.* $att^{+}{:}V_i \in C_i$ *and* $\forall j \in [1..n]$, $att^{+}{:}V_j \in C_j$ *or* $att^{\epsilon}{:}V_j \in C_j$

For example, if C_2 is the $\mathcal{L}_3$ description:

$\{Compatibility^{*}{:}\{PC, Unix\},$
$StorageRemovableType^{\epsilon}{:}\{DAT\},$
$CompressedCapacity^{\epsilon} : \{8, 24, 32, 70\}\}$

$lcs(C_1, C_2)=$
$\{RemovableDiskDrive^{?} : \{true\},$
$CD/DVD/ReadSpeed^{?}{:}\{20x, 32x, 24x\},$
$CD/DVD/Type^{?}{:}\{CDROM, CDRW\},$
$Compatibility^{*}{:}\{MAC, PC, Unix\},$
$StorageRemovableType^{\epsilon}{:}\{SuperDisk, ZIP, JAZ, DAT\},$
$CompressedCapacity^{?}{:}\{8, 24, 32, 70\}\}$

Computing the lcs of $\mathcal{L}_3$ descriptions is linear in the number of descriptions and quadratic in their size.

3 Construction of a lattice of $\mathcal{L}_2$ classes

The goal is to structure a set of data described in $\mathcal{L}_1$ into clusters labelled by $\mathcal{L}_2$ descriptions, and organized in a lattice providing a browsable semantic interface facilitating the access to the data for end-users.

We proceed to a two-step clustering:

1. In the first step, the data are partitioned according to their type: for each type c, we create a *basic class* named c. Its set of instances, denoted $inst(c)$, is the set of data of type c. Its $\mathcal{L}_2$ description, $desc(c)$, is obtained by computing the least common subsumer of the abstractions of its instances. The result of this step is a set $\mathcal{C}$ of basic classes and a set $\mathcal{A}$ of attributes supporting the $\mathcal{L}_2$ descriptions of the classes of $\mathcal{C}$. For each attribute a, the set $classes(a)$ of basic classes having a in their description is computed. This preliminary clustering step has a linear data complexity.
2. In the second step, a lattice of clusters is constructed by gathering basic classes according to similarities of their $\mathcal{L}_2$ descriptions. In this step, clusters are unions of basic classes. The computational complexity of this step does not depend on the number of initial data but only on the size of the $\mathcal{L}_2$ descriptions of basic classes.

We now detail this second step. A cluster $c_{i_1} \ldots c_{i_k}$ will appear in the lattice if the $\mathcal{L}_2$ descriptions of the classes $c_{i_1} \ldots c_{i_k}$ are judged similar enough to gather their instances. The similarity between class descriptions is stated by attributes in common. However, we take into account only attributes that do not occur in too many (or all the) classes. For instance, the attribute *price* may appear in all the instances of a catalog describing products, and is therefore not useful to discriminate product descriptions. Among the set $\mathcal{A}$ of attributes, we select meaningful attributes as being the attributes $a \in \mathcal{A}$ such that $\frac{|classes(a)|}{|classes|} \leq s$ where s is a certain threshold (e.g., $s = 0.8$). Let $\mathcal{A}_0$ be the set of meaningful attributes. We redescribe all the basic classes in terms of the attributes of $\mathcal{A}_0$ only: for a basic class c, we call its *short description*, denoted $shortdesc(c)$, the $\mathcal{L}_2$ description of c restricted to the meaningful attributes: $shortdesc(c) = desc(c) \cap \mathcal{A}_0$.

Our clustering algorithm, $\mathcal{L}_2$-*Cluster*, is described in Algorithm 1. It is adapted from a frequent item set algorithm ([2]). It iteratively builds levels of clusters, starting with building the level of the coarsest clusters corresponding to unions of classes having atleast one attribute in common. Each iteration k is guided by attribute sets of increasing size k which, being common to some class descriptions, are the support of the creation of a potential node gathering those classes. Among those potential nodes, we effectively add to the lattice those whose $\mathcal{L}_2$ short description is equal to their k-support: the k-support of a node generated at iteration k is the k-itemset supporting the generation of that node. By doing so, we guarantee that the description of the nodes added to the lattice is strictly subsumed by those of their fathers.

Notation: We call a k-itemset a set of attributes of size k. We assume that attributes in itemsets are kept sorted in their lexicographic order. We use the notation $p[i]$ to represent the i-th attribute of the k-itemset p consisting of the attributes $p[1], \ldots, p[k]$ where $p[1] < \ldots < p[k]$.

Figure 1 shows the lattice returned by $\mathcal{L}_2$-*Cluster* when it is applied on $\mathcal{C} = \{c_1, c_2, c_3, c_4, c_5\}$ and $\mathcal{A} = \{a_1, a_2, a_3, a_4\}$ such that:

$shortdesc(c_1) = \{a_1, a_2, a_3\}$ $\quad shortdesc(c_2) = \{a_2\}$
$shortdesc(c_3) = \{a_1, a_3\}$ $\quad shortdesc(c_4) = \{a_3, a_4\}$
$shortdesc(c_5) = \{a_1, a_3\}$

The following proposition summarizes the properties of the algorithm $\mathcal{L}_2$-Cluster.

Proposition 5 (Properties of $\mathcal{L}_2$-Cluster) *Let $\mathcal{H}$ be the lattice returned by $\mathcal{L}_2$-*Cluster.

- *For each node $n \in \mathcal{H}$, let $shortdesc(n)$ and $classes(n)$ be respectively the description and the set*

Require: a set $\mathcal{A}_0$ of meaningful attributes: for each $a \in \mathcal{A}_0$, $classes(a)$ is the set of basic classes of $\mathcal{C}$ whose $\mathcal{L}_2$ *short description* contains a.

Ensure: return a lattice organized in levels of nodes. Each node n is characterized by $classes(n)$: the basic classes it gathers, and $shortdesc(n)$: the least common subsumer of the short description of the basic classes of the cluster.

(* Initialization step gathering the biggest unions of classes having atleast one attribute in common:*)
$\mathcal{A}_1 \leftarrow \mathcal{A}_0, level(1) \leftarrow \emptyset, \ldots, level(\mid \mathcal{C} \mid) \leftarrow \emptyset$
for every $a \in \mathcal{A}_1$ **do**
  let $classes(a) = \{c_1^a, \ldots, c_j^a\}$
  let $desc = lcs_{\mathcal{L}_2}(desc(c_1^a), \ldots, desc(c_j^a))$
  if $desc \cap \mathcal{A}_0 = \{a\}$ **then**
    add to $level(j)$ a node n such that:
      $classes(n) = \{c_1^a, \ldots, c_j^a\}$;
      $shortdesc(n) = desc \cap \mathcal{A}_0$;
      $node(\{a\}) = n$
$k \leftarrow 1$
(* Generation of new nodes supported by $k+1$-itemsets : *)
repeat
  for every pair $(p, q) \in \mathcal{A}_k$ **do**
    if $p[1] = q[1], \ldots, p[k-1] = q[k-1], p[k] < q[k]$ **then**
      let $newp = p \cup \{q[k]\}$, and let $\mathcal{S}_k$ be the set of k-subsets of $newp$.
      if $\mathcal{S}_k \subseteq \mathcal{A}_k$ and $classes(node(p)) \cap classes(q[k]) \neq \emptyset$ **then**
        add $newp$ to $\mathcal{A}_{k+1}$
        let $\{c_{i_1}, \ldots, c_{i_j}\}$ be $classes(node(p)) \cap classes(q[k])$
        let $desc = lcs_{\mathcal{L}_2}(desc(c_{i_1}), \ldots, desc(c_{i_j}))$
        if $desc = newp$ **then**
          add to $level(j)$ a node n such that:
            $classes(n) = \{c_{i_1}, \ldots, c_{i_j}\}$;
            $shortdesc(n) = desc$;
            $node(newp) = n$
  $k \leftarrow k + 1$
until $\mathcal{A}_k = \emptyset$
(* Creation of the lattice. For every node n, $Fathers(n)$ group the fathers of n among the nodes of greater levels:*)
Initialize $Fathers(n)$ and $AncNotFathers(n)$ to $\emptyset$ for every generated node n.
for $i = \mid \mathcal{C} \mid -1$ downto 1 **do**
  for every node $n \in level(i)$ **do**
    for $j = i + 1$ to $\mid \mathcal{C} \mid$ **do**
      for every node $m \in level(j)$ **do**
        if $classes(n) \subset classes(m)$ and $m \notin AncNotFathers(n)$ **then**
          add m to $Fathers(n)$
          add $Fathers(m) \cup AncNotFathers(m)$ to $AncNotFathers(n)$

Algorithm 1: $\mathcal{L}_2$-Cluster

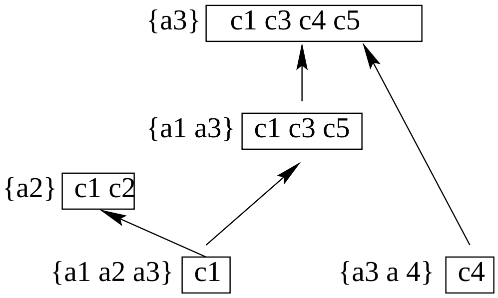

Figure 1: Example of a lattice constructed by $\mathcal{L}_2$-Cluster

of basic classes returned by $\mathcal{L}_2$-Cluster:

$$shortdesc(n) = lcs_{\mathcal{L}_2}(abst_{\mathcal{L}_2}(i_1), \ldots, abst_{\mathcal{L}_2}(i_k))$$

where $\{i_1, \ldots, i_k\} = \bigcup_{c \in classes(n)} inst(c)$.

- $\mathcal{H}$ *is a* Galois lattice, *i.e. for every node n, the pair $(classes(n), shortdesc(n))$ is maximal in the following sense: there is no $m \in \mathcal{H}$ such that $classes(n) \subset classes(m)$ and $shortdesc(n) = shortdesc(m)$, or $shortdesc(n) \prec shortdesc(m)$ and $classes(n) = classes(m)$.*
- *The worst time complexity of $\mathcal{L}_2$-Cluster is exponential in the maximal size of the basic classes $\mathcal{L}_2$ descriptions.*

4 Refinement in $\mathcal{L}_3$

The goal of this step is to refine a part of the lattice $\mathcal{H}$ computed by $\mathcal{L}_2$-*Cluster* based on the more expressive language $\mathcal{L}_3$. This step is achieved after a user chooses one node $Fatn$ and one of its descendants $Sonn$ in $\mathcal{H}$. Algorithm 2 describes how new nodes are possibly added between $Sonn$ and $Fatn$. Those new nodes correspond to clusters whose descriptions in $\mathcal{L}_2$ did not distinguish from those of $Fatn$ or $Sonn$, while having distinct descriptions in $\mathcal{L}_3$. A closure operation on those nodes is necessary in order to make their $\mathcal{L}_3$ descriptions maximal w.r.t the union of basic classes which they gather. $\mathcal{L}_3$-*Cluster* applies after the descriptions in $\mathcal{L}_3$ (denoted $desc3$ in Algorithm 2) have been computed for $Sonn$ and $Fatn$. Those computations are least common subsumer calculations whose overall time cost is polynomial w.r.t to the size and the number of the instances of the basic classes involved in $Fatn$.

Let us illustrate the application of $\mathcal{L}_3$-Cluster on the nodes $c1\, c3\, c5$ and $c1$ of Figure 1, assuming that the $\mathcal{L}_3$ descriptions of the involved basic classes are:
$desc(c_1)$=$\{att_1^+$:$\{v1, v3\}, att_2^+$:$\{v2, v4\}, att_3^\epsilon$:$\{v6\}\}$
$desc(c_3)$=$\{att_1^\epsilon$:$\{v3\}, att_2^?$:$\{v4\}, att_3^\epsilon$:$\{v7\}\}$
$desc(c_5)$=$\{att_1^\epsilon$:$\{v5\}, att_3^\epsilon$:$\{v7, v8\}\}$.
LRes-Cl and L-Cl are initialized to $\{\{c1, c3, c5\}, \{c1\}\}$.
• Gathering $c3$ with $c1$ is considered first:
$desc3$=$\{att_1^+$:$\{v1, v3\}, att_2^*$:$\{v2, v4\}, att_3^\epsilon$:$\{v6, v7\}\}$,
$classes$=$\{c1, c3\}$
Since $desc3(c5)$ is not subsumed by $desc3$, the node $c1c3$ is added. LRes-Cl becomes {{c1,c3,c5},{c1},{c1,c3}}.

```
Require: Two nodes Fatn and Sonn such that classes(Sonn) ⊂ classes(Fatn).
Ensure: return a lattice between Fatn and Sonn
L-Cl ← {classes(Fatn), classes(Sonn)}
LRes-Cl ← L-Cl
Nodes ← {Sonn}
for every node n ∈ Nodes do
  for every class c ∈ classes(Fatn) \ classes(n) do
    Change ← false
    classes ← classes(n) ∪ {c}
    if classes ∉ L-Cl then
      L-Cl ← L-Cl ∪{classes}
      desc3 ← lcs_L3(desc3(n), desc3(c))
      (* Closure operation: *)
      for every class Cl ∈ classes(Fatn) \ classes do
        if desc3(Cl) ⪯_L3 desc3 then
          add Cl to classes
          Change ← true
      if classes ∉ LRes-Cl then
        add a new node p to Nodes such that classes(p) = classes and desc3(p) = desc3
      LRes-Cl ← LRes-Cl ∪{classes(p)}
      if Change = true then
        L-Cl ← L-Cl ∪classes(p)
  Suppress n from Nodes
```

Algorithm 2: $\mathcal{L}_3$-Cluster

• Gathering $c5$ with $c1$ is now considered:
$desc3 = \{att_1^{+}:\{v1, v3, v5\}, att_2^{*}:\{v2, v4\}, att_3^{\epsilon}:\{v6, v7, v8\}\}$, $classes = \{c1, c5\}$
Since $desc3(c3)$ is subsumed by $desc3$, $c3$ is added to $classes$, which is updated to $\{c1, c3, c5\}$. The node corresponding to $c1\,c5$ is not added since it is not closed, the node corrresponding to its closure $c1\,c3\,c5$ is not added either because $\{c1, c3, c5\}$ is already in LRes-Cl.

The following proposition summarizes the main properties of the algorithm $\mathcal{L}_3$-*Cluster*.

Proposition 6 (Properties of $\mathcal{L}_3$-Cluster) *Let $\mathcal{C}_1$ be the set of basic classes of the father node. Let $\mathcal{C}_2$ ($\mathcal{C}_2 \subset \mathcal{C}_1$) be the set of basic classes of the son node. Let $\mathcal{H}_3$ be the lattice returned by* $\mathcal{L}_3$-Cluster.

- *For each node $n \in \mathcal{H}_3$, let $desc3(n)$ and $classes(n)$ be respectively the description and the set of basic classes returned by $\mathcal{L}_3$-Cluster:*

  $$desc3(n) = lcs_{\mathcal{L}_3}(abst_{\mathcal{L}_3}(i_1), \ldots, abst_{\mathcal{L}_3}(i_k))$$

  where $\{i_1, \ldots, i_k\} = \bigcup_{c \in classes(n)} inst(c)$.

- *$\mathcal{H}_3$ is a* complete Galois lattice, *i.e. for every node n, the pair $(classes(n), desc3(n))$ is maximal, and $\mathcal{H}_3$ contains every node verifying the maximality criteria and whose set of classes includes $\mathcal{C}_2$ and are included in $\mathcal{C}_1$.*

- *The worst time complexity of* $\mathcal{L}_3$-Cluster *is exponential w.r.t* $\mid \mathcal{C}_1 \mid - \mid \mathcal{C}_2 \mid$.

5 Conclusion and Discussion

This paper has proposed an approach to organize into clusters large sets of semistructured data. The scaling up of the approach is made possible because its complexity is remained in control in different ways: (1) the data are aggregated into basic classes and the clustering applies on the set of those basic classes instead of applying on the data set (2) the two-step clustering method first builds a coarse hierarchy, based on a simple language for describing the clusters, and uses a more elaborate language for refining a small subpart of the hierarchy delimited by two nodes.

Experimental results: We have evaluated our approach using a real dataset composed of 2274 computer products extracted from the C/Net catalog. Each product is described using a subset of 234 attributes, possibly multi-valued. There are 59 types of products and each product is labelled by one and only one type. The goal of the experiment was twofold : to assess the efficiency and the simplicity of the lattice for the first clustering step and to show the accuracy of the refinement of a part of the lattice using the second clustering step.

In order to make the $\mathcal{L}_2$ lattice even simpler, the number of nodes obtained with $\mathcal{L}_2$-Cluster may be parametrized by a threshold n used to restrict the nodes that appear in the lattice to gather at least n basic classes. Figure 2 shows that, as it is mentionned in [15], this quantitative criteria allows us to significantly decrease the size of the lattice.

Threshold	1	2	3	4	5
Number of nodes	119	60	24	13	9
Maximal depth	4	3	2	1	1
Running Times (s)	12.3	10.4	2.1	1.1	1

Figure 2: Quantitative results of $\mathcal{L}_2$-Cluster

Figure 3 illustrates the simplicity of the $\mathcal{L}_2$ descriptions and the significance of the nodes.

$\mathcal{L}_3$-Cluster allows to distinguish nodes that cannot be distinguished by $\mathcal{L}_2$-Cluster. For instance, if $\mathcal{L}_3$-Cluster is applied when an end-user chooses to refine the $\mathcal{L}_2$ lattice between the node (a) and the node (b) in Figure 3, the aggregation of all types of drivers (i.e. *RemovableTapeDrive*, *RemovableDiskDrive* and *HardDiskDrive*) is part of the $\mathcal{L}_3$ lattice. This new cluster appears for the following reasons : no driver is described using the attributes *StorageController/RAIDLevel*, *Networking/DataLinkProtocol* or *Networking/Type* (those attributes were optional in $\mathcal{L}_3$ description of (a)). In addition, the value *SCSI* for the attribute *StorageController/Type* is not possible for a driver.

Related work: Our work can be compared with existing work in machine learning based on more expressive languages than propositional language and/or using a

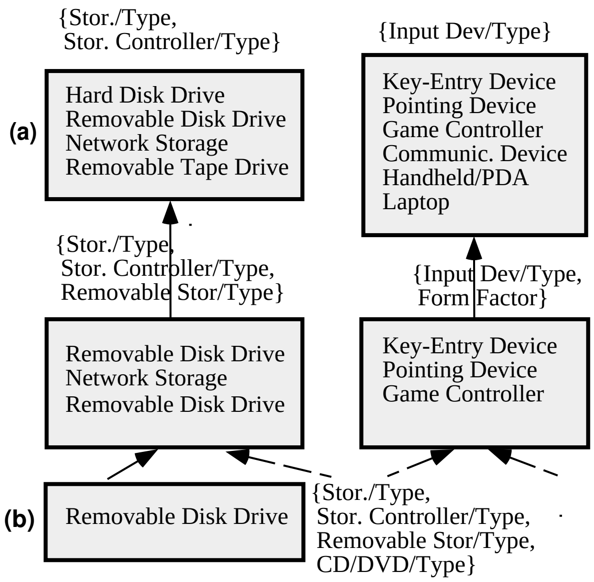

Figure 3: A part of the $\mathcal{L}_2$ lattice for C/net (n=3)

shift of representation. Most work on expressive languages has been developped in a supervised setting (e.g. Inductive Logic Programming), while little work exists in an unsupervised setting. We can cite KBG [4], TIC [5] and [11] which perform clustering in a relational setting. The main difference with our approach is that they use a distance as a numerical estimation of similarity. Although the best representation of a cluster is the least common subsumer of its instances, they approximate it numerically by the cluster centroid (i.e., the point that minimizes the sum of squared distances). The reason is that, in contrast with our setting where the lcs computation in $\mathcal{L}_3$ is polynomial, lcs computing in their first-order language may be exponential.[3] presents lcs computing for a more expressive language than $\mathcal{L}_3$ but for this language subsumption is exponential. KLUSTER [10] refines a basic taxonomy of concepts in the setting of a description logic for which computing lcs is polynomial. In KLUSTER, the clusters are not unions but subconcepts of primitive concepts, and the refinement aims at learning discriminating definitions of mutually disjoint subconcepts of a same concept. As for the use of a shift of representation, it is used in supervised learning in order to increase accuracy (i.e. the proportion of correctly predicted concepts in a set of test examples) [8; 14] or to search efficiently a reduced space of concepts [9; 6]. In unsupervised learning, shift of representational bias may be used to change the point of view about the data [12; 13]. For instance, Cluster/2 [12] provides a user with a set of parameters about his preferences on the concepts to be created. Finally, the two-step clustering approach proposed in [1] is similar in spirit with our clustering in $\mathcal{L}_2$ since it first identifies basic clusters (as high density clusters) before building more general clusters that are unions of those basic clusters.

Perspectives: We plan to extend our current work to take nested attributes and textual values into account in $\mathcal{L}_3$ in order to fully deal with XML data.

References

[1] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. *SIGMOD Record*, 27:94–105, 1998.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB-94*, Santiago, Chile.

[3] F. Baader, R. Küsters and R. Molitor. Computing Least Common Subsumers in Description Logics with Existencial Restrictions. In *IJCAI-99*, Stockholm, Sweden.

[4] G. Bisson. Conceptual clustering in a first order logic representation. In *ECAI-92*, Vienne, Austria.

[5] H. Blockeel, L. De Raedt, and J. Ramon. Top-down induction of clustering trees. In *ICML-98*, Morgan Kaufmann, 1998.

[6] P. Brézellec and H. Soldano. Tabata: a learning algorithm performing a bidirectional search in a reduced search space using a tabu strategy. In *ECAI-98*, Brighton, Angleterre, 1998.

[7] W. W. Cohen and H. Hirsh. Learning the CLASSIC description logic: Theoretical and experimental results. In *KR-94*, 1994.

[8] W. Van de Velde. Learning through progressive refinement. In Pitman, editor, *Proceedings of the third European Working Session on Learning (EWSL'98)*,London, 1983.

[9] J. Ganascia. Tdis: an algebraic formalization. In *IJCAI-93*,Vancouver, Canada, 1993.

[10] J. U. Kietz and K. Morik. A polynomial approach to the constructive induction of structural knowledge. *Machine Learning*, 14(2):193–217, 1994.

[11] Mathias Kirsten and Stephan Wrobel. Extending k-means clustering to first-order representations. In J. Cussens and A. Frish, editors, *Proc. of the 10th International Conference on Inductive Logic Programming*, Springer Verlag, 2000.

[12] R.S. Michalski and R.E. Stepp. Learning from observation: Conceptual clustering. In Morgan Kaufmann, editor, *Machine learning : An artificial intelligence approach*, 1983.

[13] G. Stumme. Local Scaling in Conceptual data Systems. In Springer Verlag, editor, *ICCS*, LNAI 1115, 1996.

[14] P. E. Utgoff. Shift of bias for inductive concept learning. In Morgan Kaufmann,ed., *Machine learning : An artificial intelligence approach Vol. 2*, 1986.

[15] R. Wille. Restructuring lattice theory. In *Symposium on Ordered Sets*, University of Calgary, Boston, 1982.