Abstractions for Knowledge Organization of Relational Descriptions

Isabelle Bournaud[1], Mélanie Courtine[2], Jean-Daniel Zucker[2]

[1] LRI, Université Paris-Sud, Av. du Général de Gaulle, 91405 Orsay Cedex, France
Isabelle.Bournaud@lri.fr
[2] LIP6 – Pole IA, Université Paris VI, 4 place Jussieu, 75252 Paris Cedex, France
{Melanie.Courtine, Jean-Daniel.Zucker}@lip6.fr

Abstract. The goal of conceptual clustering is to construct a hierarchy of concepts which cluster objects based on their similarities. Knowledge organization aims at generating the set of maximally specific concepts for all possible classifications: the Generalization Space. Our research focuses on the organization of relational data represented using conceptual graphs. Unfortunately, the generalization of relational descriptions necessary to build the Generalization Space leads to a combinatorial explosion. This paper proposes to *incrementally* introduce the relations by using a sequence of languages that are more and more expressive. The algorithm proposed, called KIDS, is based upon an iterative reformulation of the objects descriptions. Initially represented as conceptual graphs, they are reformulated into abstract objects represented as <attribute, value> pairs. This representation allows us to use an efficient propositional knowledge organization algorithm. Experiments on Chinese character databases show the interest of using KIDS to build organizations of relational concepts.

Keywords: Relational data, Unsupervised learning, Reformulation

1 Introduction

In Artificial Intelligence, the problem of automatically constructing classifications has been the subject of much research during the last fifteen years [21], [11], [14]. It consists in searching for similarities among objects that are not pre-classified and structuring them in a hierarchy in which *similar* objects are clustered. Most of the existing *Conceptual Clustering* approaches have defined this task as the search for *a* classification that would best predict unknown features of new objects [10], [12], [14]. This type of construction is guided by heuristics, which allow one to choose the best concepts among the set of possible ones. The developed methods have proved their interest in various fields [21], [11], [14], [17]. More recent research concerns the construction of classifications that *organize* knowledge [23], [4]. In this task, the goal is not to build a subset of the possible concepts but all the concepts clustering similar objects: the *Generalization Space*. In these methods, the process of construction is not based on a numerical distance among descriptions and on a function to be optimized but on a *generalization language*.

Efficient algorithms have been proposed for organizing data described by a set of pairs <attribute, value> [23] and for taking into account domain knowledge [4], [1]. Our research concerns the organization of relational data, i.e. data represented in more expressive formalisms (first-order logic, description logic, conceptual graphs). Unfortunately, the generalization of relational descriptions needed to build the Generalization Space requires matching graphs and leads to a combinatorial explosion. To help deal with the complexity of this problem, we suggest an approach that gradually increases the potential complexity of the descriptions. This is done by using a sequence of generalization languages with growing expressiveness [8], i.e. less and less abstract languages. The proposed approach, called KIDS, extends the propositional approach of knowledge organization COING [1] to a relational framework. Given a set of objects described using conceptual graphs [25] and domain knowledge represented in a generalization lattice [22], COING builds the Generalization Space of propositional descriptions of the objects. KIDS gradually enriches this space thanks to a generalization language which is made more and more expressive at each step of the algorithm. This idea, inspired from the REMO system [28], consists in increasing gradually the structure of matching. The KIDS algorithm is based upon an iterative reformulation of the data, which allows us to use COING on the reformulated descriptions of the objects.

In the next section, we present the COING method for knowledge organization. Although COING is based on relational descriptions of data, it does not use the structure of the descriptions in the construction of the Generalization Space. Section 3 introduces the notion of abstract relations and their associated Generalization Space. Section 4 describes the principle of our approach which is based upon successive graph reformulations into abstract arcs. We illustrate our approach on an example. In the next section, we evaluate KIDS on a Chinese characters database. These experiments show the feasibility of the proposed approach. Finally, in section 6, we conclude with a brief summary and outline directions for future research.

2 Organization of Relational Knowledge

2.1 A Graphical Representation of Relational Data

In the automatic construction of classifications, choosing the right language for representing the objects is very important; it has an impact on the efficiency of the algorithms manipulating them. The more expressive a language is, the more complex are the algorithms manipulating it. Objects are structured, and this is true in many fields; they may be decomposed into several parts, and these are then linked together thanks to various relations (for example a *part-of* relation). Attribute-value languages do not easily allow such structures to be represented. We use a language based on a higher-order logic and represent relational descriptions of objects in the conceptual graphs formalism. However, the choice of the conceptual graph formalism is not a limitation of our approach since it may be applied to any relational data described by graphs.

A *conceptual arc* is a triplet: `[concept`$_s$`]->(relation)->[concept`$_d$`]`, where `(relation)` corresponds to a relation between `[concept`$_s$`]` and `[concept`$_d$`]`. A *conceptual graph* is a graph composed of a set of conceptual arcs. For more information about conceptual graphs, the reader should refer to [25] [5].

Figure 1 below presents an example of a house description using conceptual graphs. The triplet `[Window]-> (colour) -> [White]` is a conceptual arc. This example is used throughout the article to illustrate the algorithms presented.

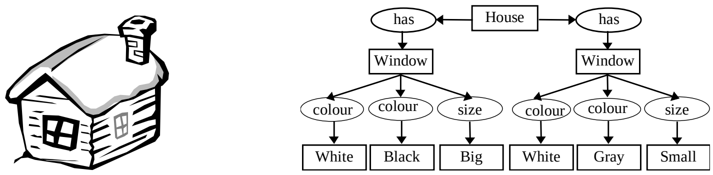

Fig. 1. A house and its description as a conceptual graph.

2.2 Organizing Data in a Generalization Space

Given a set of object descriptions and a generalization language, the associated *Generalization Space* (GS) is the set of the maximally specific conjunctive concepts generalizing these descriptions. In the GS, a node n_i is a pair (c_i, d_i). The element c_i, called the *coverage* of n_i, is the set of objects covered by n_i; and d_i, called the *description* of n_i, corresponds to the common features (least general generalization) of the objects of c_i. In the GS, a node corresponds to a cluster of objects described in *intension* by its description d_i and in *extension* by its coverage c_i. Nodes of GS are partially ordered by a subsumption relation between concepts. Given a node n_i with coverage c_i, its ancestors are all the nodes n_j, such that $C_j \supset C_i$. This partial order provided the GS with a *pruned lattice structure*[1], which may be represented by an *inheritance network*. Indeed, GS nodes *inherit* the descriptions of the nodes which are more general.

Figures 3 and 9 present two different Generalization Spaces of the same objects (as explained in the next section, part of their node descriptions come from the use of a generalization lattice over the types). Their differences lie in the expressiveness of the

[1] The Generalization Space may also be defined by the two isomorphic lattices: the Galois lattice of concept descriptions (partially ordered by the subsumption relation) and the lattice of objects (partially ordered by the inclusion relation) [20].

generalization language used to build the GS. In effect, given a set of object descriptions, depending on the language chosen to describe the generalizations (the node descriptions), the nodes of the associated GS will not be the same. The node n'3 in the GS of figure 9 for example does not appear in the GS of figure 3. Moreover, for a given set of objects, nodes belonging to different GS but having the same coverage may have a more or less general description. The node n'2 in the GS of figure 9 and the node n2 in figure 3 have the same coverage on objects ({h2, h3}) but the description of the node n'2 is more specific than that of n2.

2.3 COING: a Practical Approach to Building a Generalization Space

COING is an ascending method for building a GS: it relies upon the generalization of the objects descriptions. In COING, objects are represented using conceptual graphs. In order to deal with the problem of matching graphs which is known to be NP-complete [13], [16], COING transforms the graph representation into an arc representation. In other words, each graph describing an object is transformed into a set of *independent* arcs. For example, the graph describing the house in figure 1 is decomposed into a set of 8 arcs. Instead of trying to match a graph G1 with a graph G2, COING only searches for a *partial* matching of an arc from G1 with an arc from G2 [1]. This restriction has been previously used in [23]. It has the advantage of limiting the complexity of the algorithm (in the worst case quadratic with the number of objects [1]) because, as the arcs are oriented they fully match [2]. However, this restricts the generalization language since the relations among arcs are not considered.

The COING principle for building the GS is as follows:

1. *Reformulate* each graph describing the objects to be organized as a set of arcs.
2. *Generalize* each arc describing the objects. COING integrates an efficient method for taking into account domain knowledge in the GS construction [2]. This knowledge, represented in a generalization hierarchy (called the "type lattice" in the conceptual graphs formalism [25]) expresses, for example in the domain of colours, that the type `Black and White` (noted `B&W`) is a generalization of the three types `White`, `Black` and `Gray`. Figure 2 below presents part of the concept type lattice used for the houses.
3. *Group* the generalized arcs and initial arcs covering the same set of objects. For example, the arc `[Window]->(colour)->[B&W]` is a generalization of the two arcs (thanks to the type lattice above on figure 2): `[Window]->(colour)->[Gray]` and `[Window]->(colour)->[White]`. This arc will be part of the description of the node covering objects described by one of these arcs.
4. *Filter* the generalized arcs. Indeed, for a given matching there are several possible generalizations. For example, the two arcs `[Window]->(colour)->[B&W]` and `[Window]->(colour)->[Colour]` are both generalization of the arcs: `[Window]->(colour)->[White]` and

`[Window]->(colour)->[Gray].` This step considers each set of arcs for a node and chooses the arcs that will form the description of this node in the GS. In constructing the GS, the number of generalizations is limited while considering only the most specific ones. The filtering step thus consists in memorizing only the most specific arcs (on the example above, the arc `[Window]->(colour)->[B&W]`).

5. Finally, the nodes are *connected* thanks to the inclusion relation existing among their coverages.

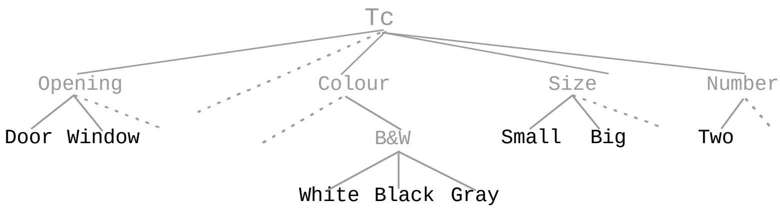

Fig. 2. Part of the concept type lattice used for the house domain

In order to illustrate the COING approach, let us consider the three houses h1, h2 and h3 whose descriptions need to be organized. These houses are described by their windows which have two proprieties: a colour and a size. Figure 3 below presents the GS built by COING for these houses.

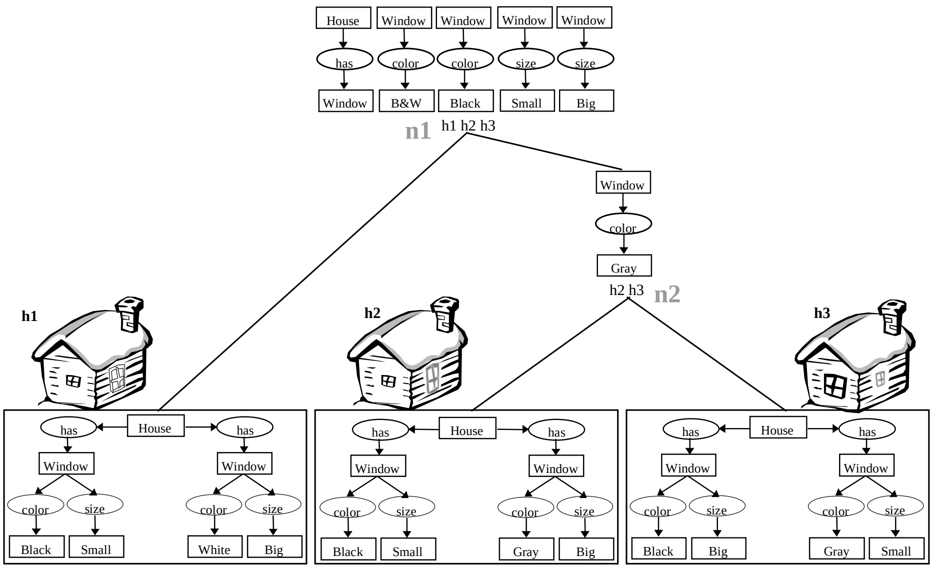

Fig. 3. Generalization Space built by COING

This Generalization Space contains two class nodes (n1 and n2) and three object nodes corresponding to the houses (box nodes). The node n2, for example, clusters the houses h2 and h3. Its coverage is {h2, h3} and its description is the arc `[Window]->(colour)->[Gray].` This class node indicates that h2 and h3 have at least a `gray` window in common in their descriptions and that this property is not shared by any other object considered. Thanks to the structure of the GS, we may add the description of the root node (n1) to this description. More precisely, we add the arcs from n1 which are not generalizations of arcs from n2, for example the arc `[Window]->(Size)->[Big].` Finally, the GS indicates that the two houses h2 and h3 have `window(s)`, which have a size (`Small, Big`) and a colour (`gray` and `black`).

Let us clarify why the arc `[Window]->(colour)->[B&W]` appears in the root node and why the arc `[Window]->(size)->[Size]` does not. This explanation will clarify the 3rd step of the COING principle (cf. previous page).

- The arc `[Window]->(colour)->[B&W]` is a generalization of the arc `[Window]->(colour)->[Black].` As this last arc is more specific and since they have the same coverage on objects ({h1, h2, h3}), the arc `[Window]->(colour)->[B&W]` should not appear. However, this arc is *useful* because its coverage on arcs is bigger than that of `[Window]->(colour)->[Black]:` it also covers the arcs `[Window]->(colour)->[White]` and `[Window]->(colour)->[Gray]`. In fact, this arc tells us that there is a window whose colour is `[B&W].`
- Consider now the arc `[Window]->(size)->[Size].` It is more general than both the arcs `[Window]->(size)->[Small]` and `[Window]->(size)->[Big]`. The coverage on objects of these three arcs is the same ({h1, h2, h3}). The coverage on arcs of `[Window]->(size)->[Size]` is exactly the union of the coverage on the arcs of the two arcs `[Window]->(size)->[Big]` and `[Window]->(size)->[Small].` The arc `[Window]->(size)->[Size]` is therefore not useful and not informative; it should not be part of the root node description.

2.4 The Trade-off between a Truly Relational Generalization Space and an Efficient Algorithm

In order to deal with the traditional knowledge representation trade-off [13] between an expressive language and an efficient algorithm, COING reformulates conceptual graphs into conceptual arcs. This simplification supplies the COING algorithm with a quadratic complexity in the number of objects, but restricts the generalization language, i.e. the expressiveness of the descriptions of the GS nodes. Let us illustrate this point using the house example. The three houses h1, h2 and h3 all have a small window and a black window; for h1 and h2 it is the same window, whereas for h3 it is not. This difference does not appear in the classification built by COING (see figure 3) since it requires representing relations between two arcs.

Given a set of objects described as graphs in the conceptual graph formalism, each node in the GS would ideally be represented by the graph that is the least general generalization of the graphs describing the objects it covers. Let us note this generalization space as GS_{max}. In fact, due to the complexity of the subsumption relation and the exponential growth of the length of the least general generalization, building GS_{max} directly using an exhaustive method is not practical. The matching curse is also true for the first-order languages used in Inductive Logic Programming (ILP) [24]. The syntactical restriction on clauses (such as ij-determinacy) used to devise efficient ILP algorithms [24] are similar to the restrictions on graphs used to devise graph-based algorithms [1], [20]. Informally, most of them consider a generalization language that avoids matching graphs as much as possible. One of the systems that performs extensive (sub-)graph matching is SUBDUE [7] whose goal is to discover substructures in data using a fuzzy graph match. However it is concerned with repeated sub-graphs within *one* given object and does not address the problem of building a GS. Liquière and Sallantin [20] proposed an algorithm that directly builds a GS but the graphs considered should be *locally injective*.

Apparently, the matching curse is such that efficiently building a Generalization Space means ignoring the inherent complexity of one-to-many relations that are the source of the matching complexity. The solution proposed in this paper is to build an initial GS using a propositional language and then to iteratively enrich the descriptions of its nodes.

3 Abstract Relations and Abstract Generalization Spaces

As stated informally by Giunchiglia and Walsh [15] an abstraction "*is the process of mapping a representation problem called "the ground" representation onto a new representation, called the "abstract" representation which helps deal with the problem in the original search space preserving certain desirable properties and is* ***simpler*** *to handle as it is constructed from the ground representation by "throwing away details"*". In fact, the reformulation step of the COING algorithm presented in section 2 may be seen as a particular abstraction [6], [9]; an abstraction of graphs that "*threw away*" details about relations between arcs. This abstraction[2] has proven to be very efficient in terms of complexity for building the Generalization Space but is a restriction of the generalization language as mentioned above. Let us call GS_0 the obtained generalization space. This section explores other related abstractions and their use to iteratively enrich the Generalization Space GS_0.

3.1 Enriching the Description of the Nodes of GS

Let us first state a property with respect to Generalization Spaces:

> *If there exists a sub-graph S_{gn} which generalizes n object descriptions, then there is in GS_0 a node whose coverage contains these n objects (and possibly*

[2] Using Giunchiglia and Walsh terminology, this abstraction is a TD-abstraction [15].

others) and whose description contains all the arcs of the generalizing sub-graph S_{gn}.

In other words, this property of GS means that to enrich any node of a GS, it is sufficient to restrict the search for richer descriptions only to the objects it covers. This principle simplifies the process of enriching a GS. In effect, the nodes of GS_0 (found by COING) are a subset of the nodes of a GS whose generalization language is richer than the one used in COING and the description of each node of GS_0 is more general than that of GS.

3.2 Abstract Relations

Underlying the approach proposed here is *a view of particular sub-graphs as abstract arcs*. Informally, given two concepts `concept`$_s$ and `concept`$_t$ in a graph, an *abstract relation* `relation`$_a$ corresponds to a relation between `concept`$_s$ and `concept`$_t$ denoted by the path between them. Let us consider the following sub-graph made of two connected arcs:

```
[House] -> (has) -> [Window] -> (size) -> [Small]
```

The triplet `(has)->[Window]->(size),` which is in the box, may be abstracted into a new relation `(has-size)`$_a$:

```
[House] -> (has-size)a -> [Small]
```

The obtained arc (containing an abstract relation) is called an *abstract arc*. Given a set of abstract relations, a graph G_a describing an object may be reformulated into a set of abstract arcs. Since the graph G_a is represented as arcs, COING may be used to organize it.

3.3 Principle of an Iterative Algorithm to Enrich GS

The principle of the proposed approach is to incrementally enrich GS by gradually increasing the depth of the abstract relations used. KIDS starts with the GS provided by COING, then in the first step, it uses a language of abstract relations corresponding to two connected arcs, in the second step a language of abstract relations composed of three connected arcs, and so on. In each step, the objects descriptions corresponding to the GS nodes that may be enriched are reformulated into abstract arcs based upon these relations. The reformulated descriptions are then processed by the COING system. This approach is inspired by the supervised learning system REMO in which the structure of matching is enriched at each step [28].

The figure 4 below presents the general KIDS principle.

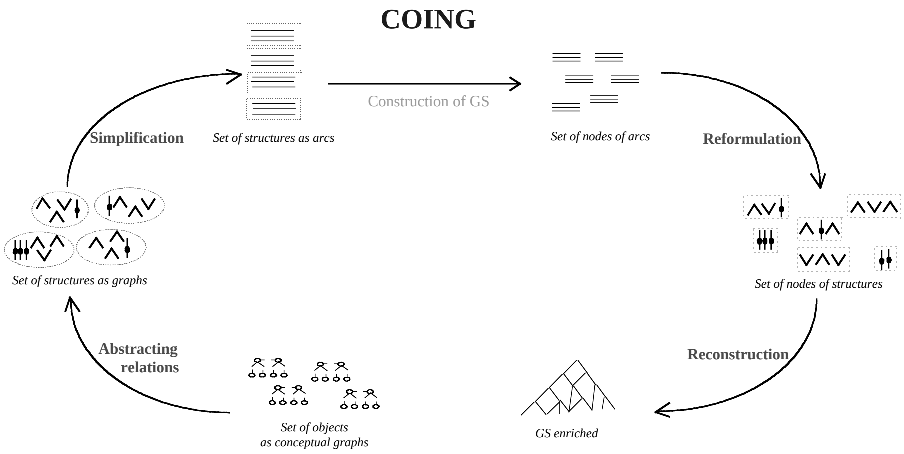

Fig. 4. Principle of KIDS.

4 KIDS: an Algorithm to Organize Conceptual Graphs

4.1 Canonical Abstracted Relations

Given the language of conceptual graphs, and depending on relations between concept types, there are numerous kinds of abstract relations. The number of reformulations of a graph into a set of abstract arcs typically increases with the number of abstract relations allowed. There is indeed a need for characterizing a set of abstract relations that will correspond to a particular generalization language. Our goal is to propose a set of abstract relations paramerized by a *level*. Abstract relations of level *l* correspond to sub-graphs consisting of *l*+1 arcs. Using an abstract relation of level *l* correspond to a generalization language of GS where *l*+1 arcs are connected; COING abstract relations (level 0) correspond to *one* arc.

We have defined the following types of sub-graphs: *sequence, star* and *hole*.

Definition 1 (*sequence-structure*): *A* ***sequence*** *is composed of a succession of arcs, which are connected one-to-another thanks to a common concept. This concept is the target of the first arc and the source of the other one.*

Fig. 5. Example of a sequence-structure composed of two arcs through the common concept of Window.

Abstract relations corresponding to sequence structures have been defined in section 3.2. In this latter case, because the sequence is an ordered structure, the direction of the abstract relation is implicitly given.

Definition 2 (*star-structure*): *A **star** is composed of a set of conceptual arcs which have the same source.*

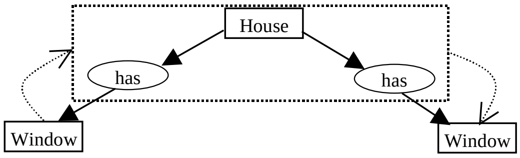

Fig. 6. Example of a star-structure composed of two arcs.

In the case of star structures, depending on the choice of the source and target concepts ($\texttt{concept}_s$ and $\texttt{concept}_t$) for the abstract relation considered, several abstract arcs may be generated. Let us consider the sub-graph of figure 6 above. The triplet `<-(has)<-[House]->(has)->` which is in the dotted box, may be abstracted into a new relation $\texttt{(two-sources-has)}_a$:

$$\texttt{[Window]->(two-sources-has)}_a\texttt{->[Window]}$$

In this particular case, both abstract arcs are identical because the concepts at stake are both `Window`. The semantics of this abstract relation may be informally given has "*targets of a same source*".

Definition 3 (*hole-structure*): *A **hole-structure** is composed of a set of conceptual arcs which have the same target (concept).*

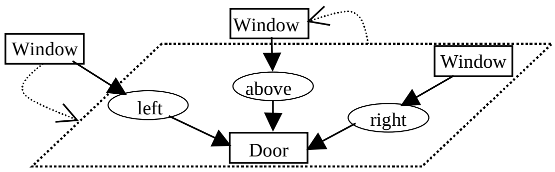

Fig. 7. Example of a hole-structure composed of three arcs.

In the case of hole structures, depending on the choice of $\texttt{concept}_s$ and $\texttt{concept}_t$, several abstract arcs may also be generated. The semantics of the abstract relation may be informally given has "*sources with the same target*". It corresponds to the inverse of a star structure. In the above example of figure 7, any two of the three `Windows` may be related by an abstract relation. One abstract relation is directly represented on figure 7 in the dotted box, where the `window(left)` is related to the `window(above)`.

`[Window] -> (three-origins-position)`$_a$ `-> [Window]`

Let us illustrate the kind of graphs KIDS manipulates at the different steps. At the first step, KIDS manipulates structures of level 1 composed of 2 arcs (cf. Figures 5 and 6). At the second step, KIDS manipulates structures of level 2 composed of three arcs (cf. Figure 7); at the i step KIDS manipulates structures of level i composed of i+1 arcs. Figure 8 below presents the structures performed by KIDS at levels 1 and 2 for the house described in figure 1. Notice that this description does not contain any hole structures, nor sequence structures of 2nd level.

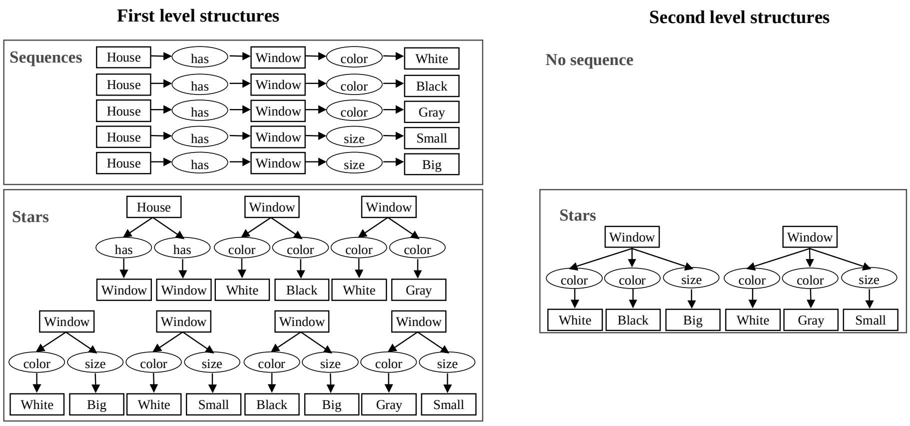

Fig. 8. Example of structures manipulated by KIDS.

The choice of these three sub-graphs (sequence, star and hole) was guided by a simplicity and generic criteria. We are currently working on formally demonstrating that any graph may be generated using these structures. One could choose other sub-graph structures as well, but they would have to respect this hypothesis in order to allow the method to truly perform any graph.

4.2 KIDS Algorithm

The principle of the KIDS algorithm is to explore, at the ith step, only the nodes which may be enriched, i.e. the nodes whose descriptions potentially contain an ith level structure. In practice, at step (i+1), KIDS explores all the nodes which were modified in step i. Indeed, a (i+1)th level structure is the aggregation of an ith structure and *one* arc. We define a candidate node as following:

Definition 4 (*candidate node*): *a node of the GS is a candidate node for KIDS at step i if it has been modified at step i - 1.*

In the first step, KIDS explores all the GS nodes built by COING. The GS enrichment algorithm is as follows (cf. Table 1):

1. For each object covered by a candidate node, determine its i^{th} level description: (i+1) connected arcs. It consists of abstracting the object descriptions using the three structures: sequence, star and hole.
2. Apply COING to the reformulated object descriptions. The result is the addition of new nodes to the GS and/or the modification of the descriptions of existing GS nodes. Notice that the new descriptions found by COING have to be reformulated in terms of sub-graphs. It consists of reformulating the descriptions using the abstract relations.
3. If KIDS modifies the GS at the i^{th} step, then repeat the method from 1) at the $(i+1)^{th}$ level (i+2 connected arcs).

```
KIDS_Algorithm (GS: Generalization Space; l: level)
GS_modified ← false
Nodes_List ← list of GS candidate nodes
for all the nodes n of Nodes_ List do
   Objects_ List ← Description of n's objects at the lth level
   GS_enriched ← COING_Algorithm(Objects _List)
   if GS_enriched modified then GS_modified ← true
   GS ← Add (GS_enriched, GS)
end for
if GS_modified == true then KIDS_ Algorithm(GS,l+1)
```

Table 1: KIDS main algorithm

While the complexity of the matching for generalization is avoided by the use of abstract relations, the complexity of graph matching is not suppressed; it is instead moved to the reformulation of the descriptions. In fact, the more complex the abstract relations are (the higher the KIDS level), the more complex the reformulation is. Nevertheless, the GS's specific structure and KIDS's iterative method allow us to limit the number of nodes to be explored at each step, while exploring only the ones that can be enriched.

However, in order to find all the structural similarities among the descriptions, KIDS needs to be applied up to the level of structure of the maximum level in the objects descriptions. In other words, if there are at least two descriptions including a structure of level l, KIDS will have to be applied up to the l level to assure a search for all the similarities.

KIDS stops either when there is no more candidates node, or when it is not possible to describe the objects at the next level (there is no structures of (i+2) arcs in the descriptions). Experimentally, the time needed to apply the algorithm at the next level may be evaluated from the time needed to build the GS at the previous level. It is possible to approximate the time required for the next level and to stop KIDS if this time is too long. Experiments in section 5 show that in our particular domain, the increase of time required between two successive levels is linear. As such, KIDS may be seen as an “anytime algorithm”. Anytime algorithms give intelligent systems the capability to trade off deliberation time for quality of results [27].

4.3 Example: Organizing Relational Data with KIDS

Let us consider again the example of the houses presented in section 2.3 (figure 3) to illustrate KIDS's improvement over COING. Figure 9 below presents the enriched GS obtained by KIDS at the 1st step using the type lattice of figure 2. The information drawn in black is the KIDS's result and COING's are in gray.

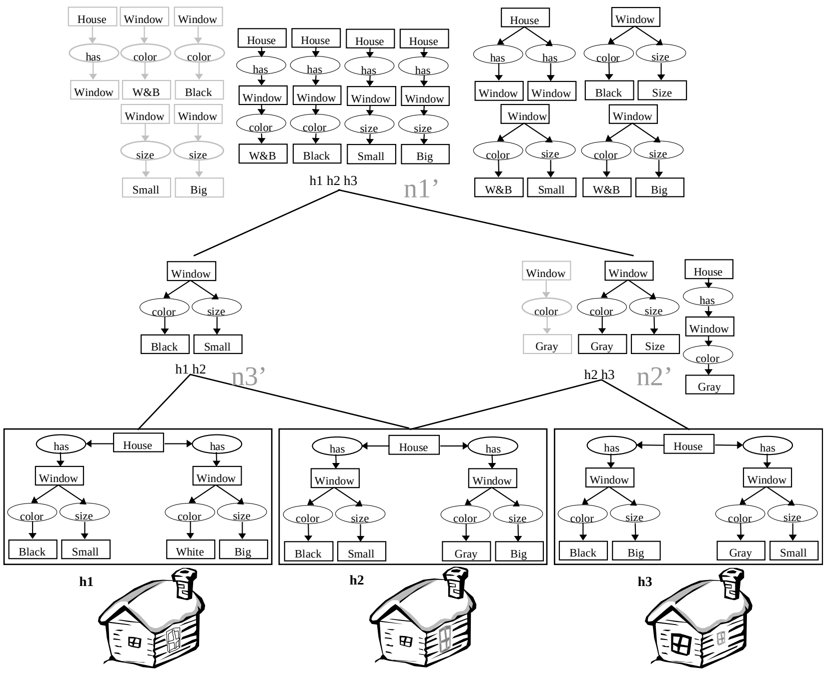

Fig. 9. Generalization Space enriched by KIDS

The abstraction allows us to discover common substructures between the object descriptions. At the 1st step, KIDS finds structural descriptions not found by COING; for example, the fact that all the houses have (at least) two `windows` and that all these windows have a color (`W&B` or `Black`) and a size (`Small` or `Big`). Furthermore, KIDS found a class clustering h1 and h2 and only these two houses: they have a `small black window` in common and this window does not appear in the description of h3 (even if h3 has a `small window` and a `black window` but it is not the same `window`). This similarity is found by KIDS at the 1st step, because it is a special composition of two arcs. In this example, KIDS enriched the description of the GS's existing nodes and added a new node to the GS clustering h1 and h2. Thus, from a GS built using a propositional language, KIDS has been able to give more precise information on the existing similarities among the objects thanks to an abstraction of sub-graphs.

For this example, it is useless to apply KIDS at the 2nd step. Indeed, the descriptions of the houses h1, h2 and h3 do not contain stars, holes or sequences of level 2; the structures contained all connect 2 arcs. Therefore, 1st level structures are enough to entirely describe the given houses h1, h2 and h3.

5 Experiments

The KIDS algorithm presented in the previous section was primarily designed to address the tradeoff between the complexity of building a Generalization Space and the expressiveness of the generalization language used to describe its nodes (cf. section 2.4). As opposed to direct methods that must first define a generalization language before building a Generalization Space, KIDS iteratively reformulates the descriptions into richer generalization languages to enrich a GS, as long as computational resources are available and enrichment is possible. This section presents several experiments aimed at evaluating KIDS empirically:

- A first dimension of experimentation considers the time complexity required to build the Generalization Space in function of the KIDS level. Indeed, in the worst case (where the size of the GS is 2^N and where nodes may be enriched at each step), the complexity of enriching the GS is, in theory, growing exponentially with the level (there is indeed no "free-lunch"). However, in practice, the GS are often much smaller, and fewer and fewer nodes are effectively enriched. In particular, our goal was to analyze the complexity in terms of both the level and the number of considered objects.
- A second dimension of experimentation concerns the evaluation of the multiplicative factor in the time required to build a GS from one level to another.
- Besides these two dimensions, we were looking for the gain of KIDS over COING in terms of the number of nodes created. Indeed, a reasonable growth of the number of nodes along with the level would account for the GS enrichment.

The domain considered for these experiments is related to the task of building classifications of Chinese characters for pedagogical purposes. We briefly recall the context of this work, the reader should refer to [1], [3] for more information about this application.

5.1 Description of the Relational Data

The database considered is a collection of 6780 Chinese characters. Each character is represented by a conceptual graph. Characters are described by the following characteristics: their initial and final pronunciation, the tone of this pronunciation, the components (between 1 and 5) and their relative positions and the key component. For example, the conceptual graph of figure 10 represents the character 情, which is composed of the radicals C5381 and C2843, which is pronounced « qing » (using the transliteration system called *pinyin*), in tone 2 and which means "feeling".

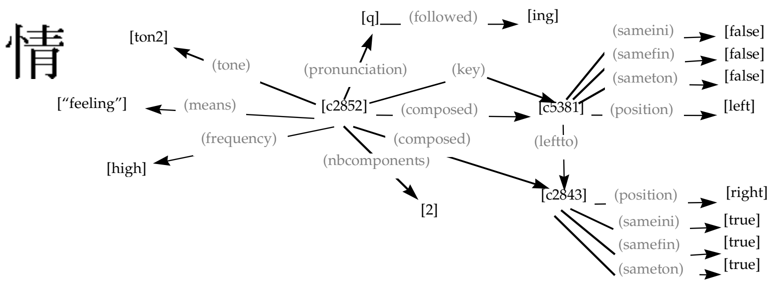

Fig. 10. Conceptual graph describing the character 情

The type lattices used for the Chinese characters are presented in the following figure 11:

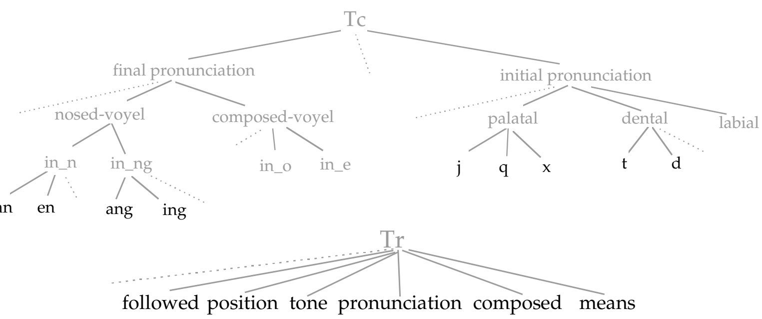

Fig. 11. Part of the type lattices for the Chinese characters

5.2 Results and Discussion

We evaluated KIDS on several databases of characters composed of 10 to 160 or 416 characters. Figure 12 shows the total time required for generating the GS for these databases using the COING and the KIDS algorithms.

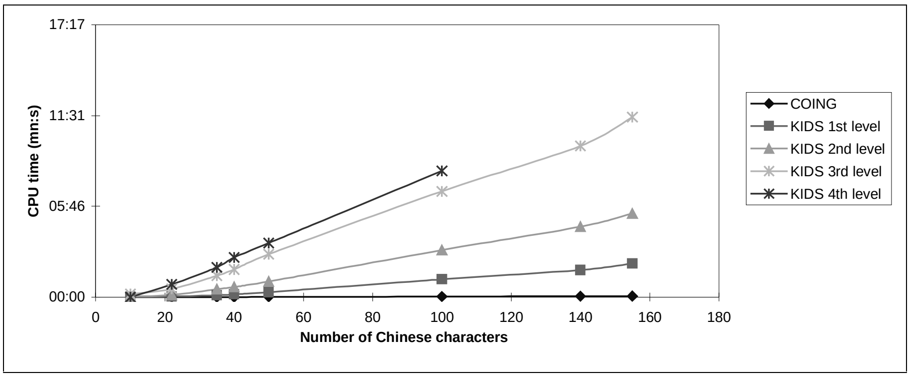

Fig. 12. Average execution time of COING and KIDS on Chinese characters *databases.*

In practice, the CPU time of the proposed algorithms is linear (it is quadratic in the worst case in COING [1]) with the number of objects. These results may be surprising because as it manipulates sub-graphs, KIDS introduces a complexity factor. However, the combinatorial explosion due to the generalization of sub-graphs is limited since a higher levels (i.e. the more complex are the graphs to be generalized), there are fewer sub-graphs matches that need to be performed.

The level introduces a multiplicative factor. The linear growth means that on the average, the time needed to move to the next level is very close to constant. Figure 13 below illustrates this result.

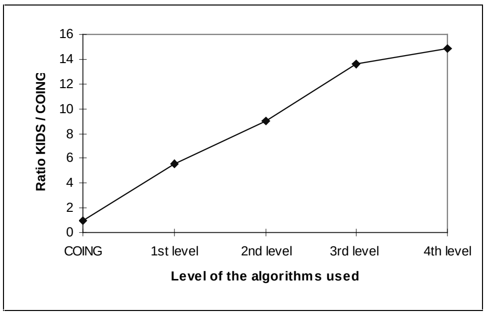

Fig. 13. Evolution of the multiplicative factor as a function of the algorithms used.

During these experiments, we also evaluated the evolution of the number of GS nodes as a function of the algorithms used. For COING, this number is in the worst case in O(N) [1]. Figure 14 summarizes these results.

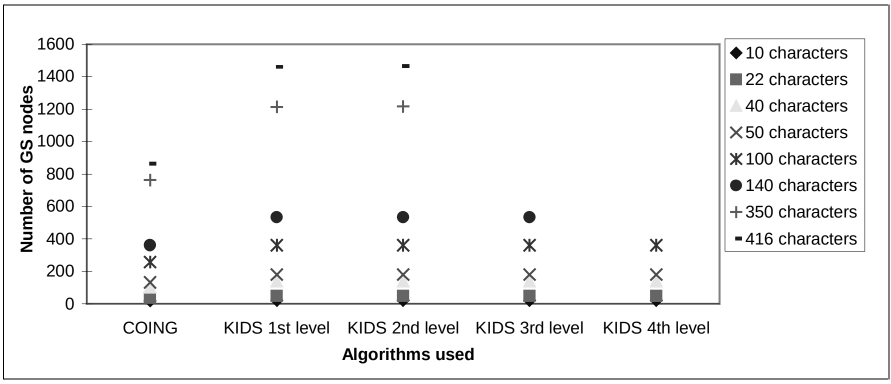

Fig.14. Evolution of the number of GS nodes

This graph shows that the number of GS nodes grows until a specific level – 1st level for the small bases and 2nd level for largest – then it becomes constant. This may be explained by the fact that from a specific level, KIDS does not allow the creation of new classes, but only enriches the already existing ones with more complex descriptions.

6 Conclusion

Organizing relational data has many applications in the field of data mining, knowledge indexation or systematic but may also be used to extract conceptual hierarchies. The problem of conceiving efficient algorithms for this purpose is hard because of the known graph-matching complexity. This paper has proposed an abstraction based approach to incrementally introduce the complexity of the relations by using a sequence of languages that are less and less abstract. The algorithm KIDS builds an initial GS using a propositional language and then iteratively enriches the descriptions of its nodes.

We have implemented and successfully tested our approach. Our experiments suggest that the proposed method provides an organization of relational concepts while keeping a linear complexity in practice with the number of objects. This result is due to the fact that the more complex the structures are, the less nodes need to be explored. Our work supports the idea that iterative abstraction may be an appropriate approach to deal with the traditional representation trade-off. By increasing the expressiveness of the language, the solution is refined at each step and results from the previous step are used to reduce the complexity of the current step.

The first perspective of this work is to characterize more precisely the generalized language used in the enriched GS. Indeed, characterizing the GS's enriched language allows us to evaluate the usefulness of sub-graphs and to filter them in order to keep the useful ones.

Another possible improvement of the algorithm is to define methods for evaluating the effectiveness of KIDS for a given database. Indeed, when types in the description of a conceptual graphs database appear only once, it is not necessary to apply KIDS to this database because the decomposition does not cause a loss of information. In contrast, if a type appears several times in the descriptions (like in the houses), it is not possible to differentiate them. So, we can consider a pre-processing on the data to evaluate the maximal level of KIDS application.

Finally, we plan to extend this method for a more efficient processing of numerical data. Currently, the numerical information contained in descriptions is processed as symbols; the implicit order existing between numbers is not taken into account. A preprocessing of descriptions would make it possible to determine a hierarchy of generalization of the numerical values. The creation of new values of attributes, as is the case in constructive induction, would make it possible to better account for the similarities between descriptions [18], [26].

Acknowledgements

The authors wish to specially thank the anonymous reviewers for their constructive reviews, suggestions and help for writing the final version of this paper. We also would like to thank Lise Fontaine for her careful proofreading of the final version.

References

1. Bournaud I.: *Regroupement conceptuel pour l'organisation de connaissances*. Ph.D. Thesis, LIP6 - Pole IA, Université Paris VI. (1996).

2. Bournaud I., Ganascia J.-G.: Accounting for Domain Knowledge in the Construction of a Generalization Space. ICCS'97, Lectures Notes in AI n°1257, Springer-Verlag, pp. 446-459. (1997).

3. Bournaud I., Zucker J.-D.: Integrating Machine Learning Techniques in A Guided Discovery Tutoring Environment for Chinese Characters, *International Journal of Chinese and Oriental Languages Information*, Processing Society, 8(2). (1998).

4. Carpineto C., Romano G.: GALOIS: An order-theoretic approach to conceptual clustering. *Tenth International Conference on Machine Learning (ICML).* (1993).

5. Chein M., Mugnier M.L.: Conceptual Graphs: Fundamental Notions, *Revue d'Intelligence Artificielle*, 6(4). (1992). 365-406.

6. Choueiry B.Y., McIlraith S., Iwasaki Y., Loeser T., Neller T. Engelmore R.S., Fikes R.: Thoughts on a Practical Theory of Reformulation for Reasoning about Physical Systems, *Symposium on Abstraction, Reformulation and Approximation (SARA-98)*. (1998). 25-36.

7. Cook D. J., Holder L. B.: Substructure discovery using minimum description length and background knowledge. Journal of Artificial Intelligence Research **1**. (1994). 231-255.

8. De Raedt L., Bruynooghe M.: An overview of the interactive concept-learner and theory revisor CLINT. Inductive Logic Programming. S. Muggleton. London, Harcourt Brace Jovanovich. (1992). 63-192.

9. Ellman T.: Hill climbing in a Hierarchy of Abstraction Spaces, Rutgers University. (1993).

10. Fisher D.: Approaches to conceptual clustering. *Ninth International Joint Conference on Artificial Intelligence (IJCAI)*, Los Angeles, CA, Morgan Kaufmann. (1985).

11. Fisher D.: Knowledge Acquisition Via Incremental Conceptual Clustering. *Machine Learning: An Artificial Intelligence Approach.* R. Michalski, J. Carbonell and T. Mitchell. San Mateo, CA, Morgan Kaufmann. II. (1987). 139-172.

12. Fisher D.: Iterative Optimization and Simplification of Hierarchical Clusterings. *Journal of Artificial Intelligence Research* 4. (1996). 147-179.

13. Garey, M., D. Johnson: Computers and intractability: A guide to the theory of NP-completeness. San Fransisco, CA, W. H. Freeman. (1979).

14. Gennari J. H., Langley P., Fisher D.: Models of incremental concept formation. *Artificial Intelligence* 40-1(3). (1989). 11-61.

15. Giunchiglia, F., Walsh T.: A Theory of Abstraction, Artificial Intelligence 56(2-3). (1992). 323-390.

16. Haussler D.: Learning Conjunctive Concepts in Structural Domains. *Machine Learning (4),* (1989). 7-40.

17. Ketterlin A., Gancarski P., Korczak J.J.: Conceptual clustering in Structured databases: a Practical Approach. *Proceedings of the Knowledge Discovery in Databases KDD'95*, AAAI Press. (1995).

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

19. Levesque H.J., Brachman R.J.: A fundamental tradeoff in knowledge representation and reasoning. In R.J. Brachman, H.J. Levesque, editor, *Readings in Knowledge Representation*, Morgan Kaufmann, (1985). 41-70.

20. Liquiere M., Sallanatin J.: Structural Machine Learning with Galois Lattice and Graphs. *Fifteen International Conference on Machine Learning (ICML)*, (1998).

21. Michalski R. S., Stepp R. E.: An application of AI techniques to structuring objects into an optimal conceptual hierarchy. *Seventh International Joint Conference on Artificial Intelligence (IJCAI).* (1981)..

22. Michalski R. S.: A theory and methodology of inductive learning, *Machine Learning: An Artificial Intelligence Approach, Vol. I.* Morgan Kaufmann (1983). 83-129.

23. Mineau G., Gecsei J., Godin R.: Structuring knowledge bases using Automatic Learning. *Sixth International Conference on Data Engineering*, Los Angeles, USA.(1990).

24. Muggleton, S., Raedt L. D.: Inductive Logic Programming: Theory and Methods. Journal of Logic Programming 19(20). (1994). 629-679.

25. Sowa J. F.: *Conceptual Structures: Information Processing in Mind and Machine*, Addisson-Wesley Publishing Company. (1984).

26. Wnek J., Michalski R.: Hypothesis-driven constructive induction in AQ17-HCI: a method and experiments. *Machine Learning 14(2)*. (1994). 139-168

27. Zilberstein S.: Using Anytime Algorithms in Intelligent Systems, AI Magazine, 17(3). (1996). 73-83,

28. Zucker J.-D., Ganascia J.-G.: Changes of Representation for Efficient Learning in Structural Domains. *International Conference in Machine Learning* (ICML'96), Morgan Kaufmann. (1996).