Linköping Electronic Articles in
Computer and Information Science
Vol. 5(2000): nr 2

Simple Pre- and Post-Pruning Techniques for Large Conceptual Clustering Structures

Guy Mineau, Akshay Bissoon, and Robert Godin

Linköping University Electronic Press
Linköping, Sweden

http://www.ep.liu.se/ea/cis/2000/002/

Posted on March 8,
formally published on December 5, 2000
by Linköping University Electronic Press
581 83 Linköping, Sweden

Linköping Electronic Articles in
Computer and Information Science
ISSN 1401-9841
Series editor: Erik Sandewall

Typeset by the author using LaTeX
Formatted using étendu *style*

Recommended citation:
<Author>. <Title>. Linköping Electronic Articles in Computer and Information Science, Vol. 5(2000): nr 2. http://www.ep.liu.se/ea/cis/2000/002/. *December 5, 2000.*

This URL will also contain a link to the author's home page.

Abstract

In (Godin et al, 1995a) we proposed an incremental conceptual clustering algorithm, derived from lattice theory (Godin et al., 1995b), which is fast to compute (Mineau & Godin, 1995). This algorithm is especially useful when dealing with large data or knowledge bases, making classification structures available to large size applications like those found in industrial settings.

However, in order to be applicable on large data sets, the analysis component of the algorithm had to be simplified: the thoroguh comparison of objects normally needed to fully justify the formation of classes had to be cut down. Of course, from less analysis results classes which carry les semantics, or which should not have been formed in the first place. Consequently, some classess are useless in terms of the information needs of the applications that will later on interact with the data. Pruning techniques are thus needed to eliminate these classes and simplify the classification structure.

However, since these classification structures are huge, the pruning techniques themselves must be simle so that they can be applied in reasonable time on large classification structures. This paper presents three such techniques: one is based on the definition of constraints over the generalization language, the other two are based on discrimination metrics applied on links between classes or on the classes themselves. Because the first technique is applied before the classification structure is built, it is called the *pre-pruning* technique, while the other two are called *post-pruning* techniques.

Authors' addresses

GM, AB: Department of Computer Science
Université Laval
Quebec City, Québec
Canada, G1K 7P4
E-mail: `mineau@ift.ulaval.ca`

RG: Department of Computer Science
Université du Québec à Montréal
Montréal, Québec
Canada, H3C 3P8
E-mail: `godin.robert@uqam.ca`

1 Introduction

Conceptual clustering algorithms form classes of similar objects. In order to do so, objects are compared to one another to detect any similarity that they may convey. Based on this similarity, common generalizations of objects are produced: they justify the formation of classes, they also serve as characteristic descriptions for the classes of objects from which they were inferred.

With our classification algorithm[1], called MSG (Mineau & Godin, 1995), the generalization language is determined without comparing the objects: it is generated solely based on the description of the individual objects. It must be general enough to include common generalizations among objects. However, if too general, it may dilute the semantics that the classes would otherwise convey. In order to avoid being overly general, a validation of the generalization language could take place. To be thorough, it would imply pairwise comparisons of objects in order to ensure that the generalization language is appropriate to represent the *maximally specific* common generalizations among them. With large data (knowledge) bases, this would bring us back to the complexity problem that prompted us to develop the MSG (summarized in Section 2 below).

Since we can not guarantee that the generalization language as produced with our technique will be appropriate to find maximally specific generalizations among objects, we will therefore need pruning techniques to eliminate classes that do not carry sufficient semantics. Since our goal is to produce efficient classification algorithms for very large data sets, and since pruning techniques must be part of these algorithms, we need to devise pruning techniques which are themselves very simple to apply (otherwise we may lose the computational gains obtained by the MSG).

Section 3 below describes a pruning technique applied on the generalization language itself (the definition of constraints on the generalization process). Section 4 describes two pruning techniques applied on the resulting classification structure: one eliminating *weak links* between classes, the other one eliminating classes which do not show a satisfying distribution of their object set into their children nodes (children classes). Because the first technique is applied before the classification structure is built, it is called a *pre-pruning* technique, while the other two are called *post-pruning* techniques.

2 Conceptual clustering for large size applications: the MSG

Classification algorithms provide a characterization of the data on which they are applied. This is useful for indexing, retrieval, knowledge discovery and explanation purposes. Industrial applications often deal with data sets that are too large to be considered by traditional approaches. Consequently, some alternative techniques needed to be devised. In (Mineau & Godin, 1995) we introduced one such technique called MSG (Method of Structuring by Generalization) which proposes to divide the classification process into

[1] In this text, even though our clustering method is unsupervised, we use the terms *classification* or (conceptual) *clustering* interchangeably.

two distinct and successive phases: 1) the formation of *promising* classes, and 2) the subsequent production of characteristic descriptions, but only for the classes identified as relevant by the particular applications which will use the data. Doing so eliminates the need for the class formation process to produce a *complete* characteristic description for each class in order to justify its existence, since it may well be the case that only a portion of the resulting classes will effectively be used by the applications. Consequently, a similarity function that is less time consuming than those that compute maximally specific common generalizations can be used in the class formation phase.

As mentioned in Section 1 above, the MSG first determines a generalization language L_g that represents a subset of all possible generalizations of the objects. Then the objects are mapped onto L_g. The meeting points in L_g will be used to form and describe classes. The idea is to bound L_g so that the determination of common generalizations among objects becomes manageable in terms of complexity. In fact, the complexity of the MSG is directly proportional to the production of L_g and to the number of objects. Naturally, the actual classes found by the MSG are highly dependent on L_g. To explain how the MSG works, Section 2.1 below presents our data set; while Section 2.2 gives an example of how it would be applied on this data set.

2.1 Our Application Data Set

As our test bed we used a small knowledge base on impressionist paintings from the National Gallery of Art of Washington D.C. Each painting (from a subset of 100 paintings) was described using conceptual graphs (Sowa, 1984). A part of the description gave bibliographic notes on the painting: author, material, year, era, title, collection, etc.; some other part described the subject matter. Since we aim at developing classification methods for both data and knowledge base environments, we chose to use complex objects from the start. Each object was described by an average of 30 (binary) relations, so altogether the knowledge base contained 3000 relations.

2.1.1 The Representation of Complex Objects

A conceptual graph (CG) is made of two types of nodes: *concepts* representing objects (either concrete or abstract), and *relations* representing semantic links between concepts. Both nodes are typed; types come from a thesaurus called a *type hierarchy* T. Identifiable individuals are referred to by individual markers. I is the set of all individual markers. All concept nodes in a CG represents an individual of some type, either represented by an *individual marker* (if explicitly known) or by an existentially quantified variable (otherwise). When the variable is existentially quantified, then the * symbol, called the *generic marker*, may be used. An individual represented by a concept node is said to conform to the type of the concept. To that effect, a conformity relation :: is defined between elements of T and $I \cup \{\star\}$. We call the elements of $I \cup \{\star\}$ *referents*. A concept is thus described using a type t and a referent which conforms to t.

Arcs connect concept nodes to relation nodes in a non-ambiguous way, as described in B, the *canonical basis* of the system. B encodes the *signature* of each relation. For each relation r, B gives the number of parameters of r, their order, and the maximal type of each parameter (which are all concept nodes). By doing so, overgeneralizations are avoided. Canonical formation operators define how concepts and relations may be connected to form new graphs. Consequently, all graphs represented in a CG system are derived from $\langle T, I, ::, B \rangle$, the *canon* of the system.

2.2 Applying the MSG on our Data Set

With our application database, the MSG must generalize a set of conceptual graphs to produce L_g. Then it must map each object onto L_g to find meeting points in this space of generalizations. These meeting points will be considered as common generalizations. Each common generalization may give rise to the formation of a new class, if the class does not already exist (because of some other common generalization). Also, we may have background information validating the formation of classes (like semantic constraints for instance (Bournaud, 1996)). With our implementation of the MSG, no such background knowledge was used in the formation of the classes.

At this point we would like to point out to the reader that the generation of (part of) L_g from an object o, called the *completion* of L_g with regard to o, and the mapping of o to L_g can be done simultaneously. L_g will be complete and all objects will be mapped onto it once they are all scanned. This is an iterative process over the set of all objects. For n objects, it is thus a $O(n)$ process, as long as the completion of L_g with regard to any object is done independently of any other object. In order to keep the complexity associated with the production of the classes as low as possible, the completion of L_g with regard to any object must also minimize its associated complexity, as discussed in Section 2.2.1 below.

2.2.1 Generalizing Conceptual Graphs

There are three generalization operators that can be applied onto a single object in order to generalize it (Michalski, 1983): operator #1) *turning a constant into a variable*, i.e., replacing an individual marker by the generic marker, operator #2) *generalizing a term*, i.e., generalizing the concept type of a concept or the relation type of a relation, and operator #3) *dropping a condition*, i.e., deleting a subgraph. Each of these operations preserve the truth-value of a graph, that is the generalized graph is a logical entailment of the graph it is derived from.

In our implementation of the MSG, to generate the completion of L_g with regard to an object o, we apply these three operators in a predefined sequence. First, we use operator #3 above to translate a graph into a set of triplets of the form: $\langle concept_1, relation, concept_2 \rangle$[2] . This is done in order to end up with a simple keyword based representation of each object so that any clustering technique using the intersection of sets of descriptors as a similarity extraction procedure can be applied. Therefore, from that point on, what we propose in terms of clustering and pruning techniques applies to any type of object representation: complex objects, attribute-value pairs,

[2] We assume that all relations in a conceptual graph are binary relations since n-ary relations can be transformed into n binary relations and an additional concept.

keywords. So, from a single graph having k relation nodes, we produce k initial generalizations of the graph, the k triplets corresponding to the k binary relations of the graph[3] . After this operation, the cardinality of G_i, the set of generalizations of graph i, written G_i, is exactly k.

Then, since we wish to use set intersection as the similarity extraction procedure between descriptions of objects, we need to generalize each set of triplets so that common semantics also appears as triplets. Consequently, each of the k triplets of each set will be generalized according to operator #1 above, which will introduce additional generalizations into G_i. Because relations do not carry referents, only the two concepts can be generalized using operator #1. This results in having G_i four times larger than what it previously was, in the worst case (i.e., if no common generalizations are found at that point among the triplets describing graph i); so $G_i = 4k = O(k)$ after this step. Finally, in order to increase the similarity between sets of triplets, operator #2 is applied on all concept types (and on no relation type[4]) of all triplets in G_i. In the original version of the MSG, in order to keep the complexity of the completion of L_g as low as possible, we used only a single generalization term: the ? wild card symbol, which symbolizes *something.* Doing so would ensure that the generalized triplets would cover triplets of other graphs as well, at a minimal cost. With only two concept types for each concept of each triplet, we ended up multiplying the size of G_i by 4; so after this last operation, $G_i = 16k$, in the worst case (if no common generalizations exist among the triplets of G_i). Again, since similarity is based on common subsets, there must be more general triplets introduced in the invidual $\mathrm{G_i}$ so that common semantics can be extracted using set intersection.

Consequently, for all n objects, we have $L_g = \cup_i G_i (\forall i \in [1, n])$, which can be constructed incrementally as objects are scanned sequentially. We have $L_g = 16nk = O(nk)$, in the worst case, for k being the maximum number of triplets encountered in the individual object descriptions. It was expected and was verified empirically that the hidden constant is greatly reduced with the overlap that exists among the different $\mathrm{G_i}$ (see Section 3 below). Figure 1 shows the binary relation R, called *intersection matrix*, between L_g (created incrementally as the objects are scanned), and the set of objects O. R encodes in which description set appears what triplet. Of course, there are efficient ways to create, update and store R, but this discussion falls outside the scope of this paper. The interested reader is referred to (Mineau, 2000) for a more thorough discussion of complexity and implementation issues.

From the analysis of R, classes are formed to represent different subsets of objects that share a common description. Different techniques use different criteria to guide the creation of classes. The clustering technique that we use forms classes which conform to the following definitions:

1. a class c_i is defined as $\langle i_i, e_i \rangle$, where i_i is the intention set of c_i, and e_i is the extension set of c_i,

[3] This step is done first so that, from that point on, the following generalization steps could be applied in cases where instead of having triplets, we would have simple keyword-based descriptors or attribute-value pairs, making our method capable of handling other types of data sets (like those found in traditional machine learning literature).

[4] We found that in practice the relation type hierarchy is so shallow that generalizing relations immediately introduces overgeneralizations. Therefore, we decided to generalize concept types only.

R = L_gx 0	#1	#2	#3	.....	#n
<Painting:#12, using, Material:#oil>	x				
<Painting, using, Material:#oil>	x				
<Painting:#12, using, Material>	x				
<?, using, Material:#oil>	x				
<?, using, Material>	x				
<Painting:#12, using, ?>	x	x			
<Painting:, using, ?>	x	x	x		
<?, using, ?>	x	x	x		x
<Painting:#12, using, Canvas:#47>		x			
<Painting, using, Canvas:#47>		x	x		
<Painting:#12, using, Canvas>		x			
<Painting, using, Canvas>		x	x		x
<?, using, Canvas:#47>		x	x		
<?, using, Canvas>		x	x		x
<Painting:#84, using, Canvas:#47>			x		
<Painting:#84, using, Canvas>			x		
<Painting:#84, using, ?>			x		
.....					

Figure 1: A typical intersection matrix.

2. e_i = the set of all $o \in O$ such that $\langle t, o \rangle \in R$ for all $t \in i_i$
3. i_i = the set of all $t \in L_g$ such that $\langle t, o \rangle \in R$ for all $o \in e_i$

The set of classes C which conform to these definitions forms a lattice structure (Wille, 1982) with regard to the inclusion relation ($\subseteq$) defined over the extension sets (or intention sets) of the classes. We say that c_1 is strictly more specific than c_2, written $c_1 < c_2$ iff $e_1 \subset e_2$. Our clustering algorithm, the MSG algorithm, produces a subset of C, C', such that each element t of L_g appears in at most one class. That class c is the most general class in C (in terms of the number of objects that it covers) where t appears (in the intention set of the class). As a result, the MSG produces at most $O(L_g) = O(nk)$ classes.

Based on the $\subseteq$ relation, links between classes can be represented explicitly. Let us define the set of links of the classes in C as

$L(C) = \{\langle c_1, c_2 \rangle\}$ where $c_1, c_2 \in C$, $c_1 < c_2$, and there is no c_3 such that $c_1 < c_3 < c_2$.

The classification structure described as $\langle C, L(C) \rangle$ is called a *concept lattice* (Wille, 1982); the MSG produces a classification structure which is described as: $\langle C', L(C') \rangle$. Though we based our experiments on the use of the MSG, the pruning methods and results presented in this paper apply to any set of classes that form a partial order (with regard to the $\subseteq$ relation over their extension sets). The MSG in itself is only accessory to the argumentation made in this paper; for a more detailed presentation of the MSG itself and its related complexity, the interested reader is referred to (Mineau & Godin, 1995) and to (Godin et al., 1995a).

2.2.2 Overgeneralizations

It should be clear that since operator #2 only uses one general unifier for obvious computational reasons, this favors the formation of overly general classes. For instance, a class whose intention set would only contain: $\langle ?, \textit{using}, ? \rangle$, would represent all pairs of objects being in relation *using* with one another. This may be useful in occasional circumstances where relations may be used as indexes, but it does not help to characterize the data since the ? symbol may represent very heterogeneous objects. In fact, it does not carry sufficient semantics to be useful to semantic driven applications. However, we could use T, the hierarchical vocabulary of the system, i.e., the type hierarchy, in order to generalize concept types with more specific unifiers than the ? symbol. The idea is to search T from some concept type (the one in the triplet to generalize) towards the top of the type hierarchy. The length of the search path could be bounded by some value in order to avoid including too general unifiers, as usually found near the top of type hierarchies. Of course this additional search has a cost both in terms of storage and computing resources: it would increase the hidden constant mentioned above because the height of the type hierarchy would now play a part in the complexity of our method. However, in practice, it is often the case that type hierarchies are relatively shallow. In any case, by carefully choosing some bound on the number of types used to replace the concept types in the original triplets, we hope to increase the semantics conveyed by the classes, at a reasonable computational cost. Using intermediate concept types (those appearing between the original triplet and the top of the type hierarchy), we hope to add useful classes into the classification structure. By avoiding too general unifiers, we hope to eliminate classes which carry overly general semantics and which are thus useless to the applications. Section 3 below presents the findings of this enquiry.

3 Pre-Pruning

This section presents comparative results pertaining to the use of T, the type hierarchy of the domain, in the generalization process of our classification algorithm. In the first set of experiments, T is used as such (without modifications); in the second set of experiments, T is broken down into a set of disjoint type hierarchies $T^\star$. In T every pair of concept types may be unified through some common supertype, therefore T is called a *unified vocabulary*. In contrast, $T^\star$ is called a *fragmented vocabulary*. Figure2 below shows a subset of T[5] ; its fragmented counter-part is produced by deleting the bold types from T: Physical_Object, Artefact and Material. As mentioned in Section 2.2.2, the search for more general terms (supertypes) of a concept c in a graph will consider all supertypes from the concept type of c up a certain number of levels l in T. With a unified vocabulary, it is expected that some overly general supertypes may be introduced even with low values of l. It all depends on the initial concept type of c, i.e., on how general is c in the first place. Of course, the maximum length path for T will be greater than that of $T^\star$, leading to more general classes, and possibly to overly general classes. Consequently, it is expected that the classes produced under $T^\star$ will be more specific than those produced under T, for a lower

[5] Even though in our example the type hierarchy is a tree, in the general case there is no such restriction.

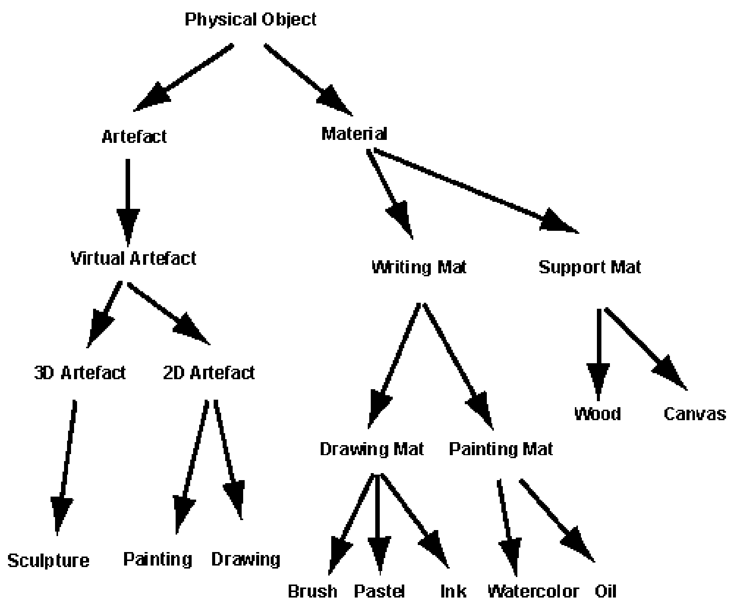

Figure 2: Part of the type hierarchy of the domain.

computational cost. However, we must point out that $T^{\star}$ requires background knowledge on maximal unifiers (generalizations) of concept types in order to be applicable. The acquisition of this information requires human intervention, which may not be possible to have: this information or the human resources to acquire it may not always be available.

In the experiments reported below, we increased the maximum path length l in order to compare the resulting classification structures in terms of standard performance metrics: number of nodes, of triplets (descriptors), of links, computing time, etc. In the subsequent figures, each classification structure is thus identified by h_l. For instance, h_0 is the classification structure produced with no generalization of the concept types; while h_1 implies that each concept type was replaced by its immediate supertypes. With our type hierarchy, the maximum length path is 7 for T and 5 for $T^{\star}$, so h_7 and h_5 are the last classification structures considered by each approach respectively. The tests reported in the following subsections were conducted on a Sun Workstation 20.0 under Solaris 2.5; the algorithms were written in C.

3.1 Number of Nodes

Figure 3 and Figure 4 show the growth in number of nodes (using T and $T^{\star}$ respectively) as the maximum path length increases. The reader should notice that in both cases, the number of nodes increases up to some point and then decreases. This was expected, as the introduction of triplets creates new nodes (classes) and eliminates older nodes. Depending on which phenomenon is more important at one point, an increase or a decrease in the total number of nodes is observed. It is normal that new triplets (generated through the generalization of some object) introduce new nodes unless these triplets were already present in the classification structure. In that case, they were previously introduced by the generalization of some other

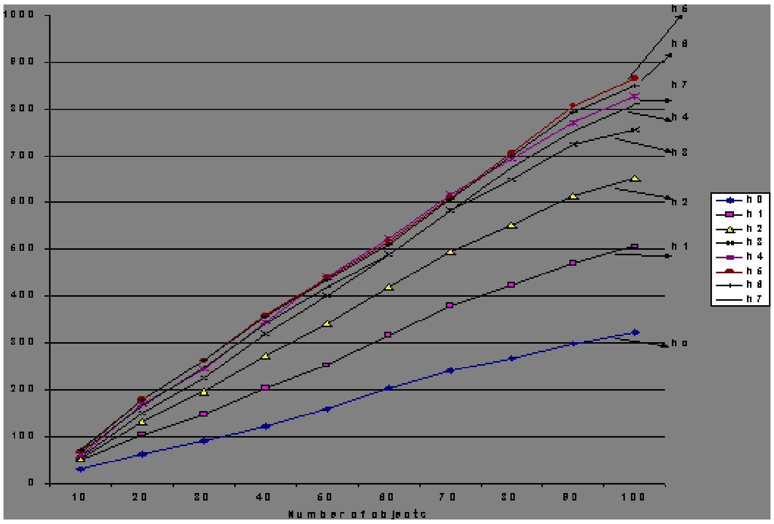

Figure 3: Number of nodes vs number of objects, under T.

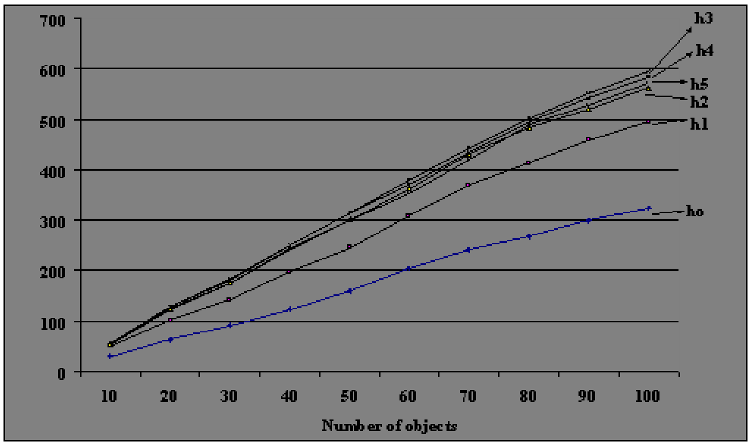

Figure 4: Number of nodes vs number of objects, under $T^{\star}$.

object. Since they now cover more objects, they migrate to more general classes. If the more specific classes from which they migrate become empty, then these nodes are eliminated. We call this phenomenon *total inclusion* of a class description by some other class description. The total number of nodes in a classification structure may decrease as a result of total inclusion.

First, we must say that the number of nodes found by the initial MSG (where only the ? symbol was used as a unique general unifier) was 750. Consequently, with the unified vocabulary we should bound l, the maximum length path, by 3; otherwise we would obtain more classes than with the original version of the MSG. Comparing h_3 of Figure 3 with h_3, h_4, or h_5 of Figure 4 (which are sensibly the same), we see that the fragmented vocabulary improves the number of classes by more than 20%. However, some of the remaining classes may still be too general since we observe the same order of improvement with l = 3, 4, or 5 independently. Obviously, other pruning techniques will be needed to further simplify the classification structure (as proposed in Section 4 below).

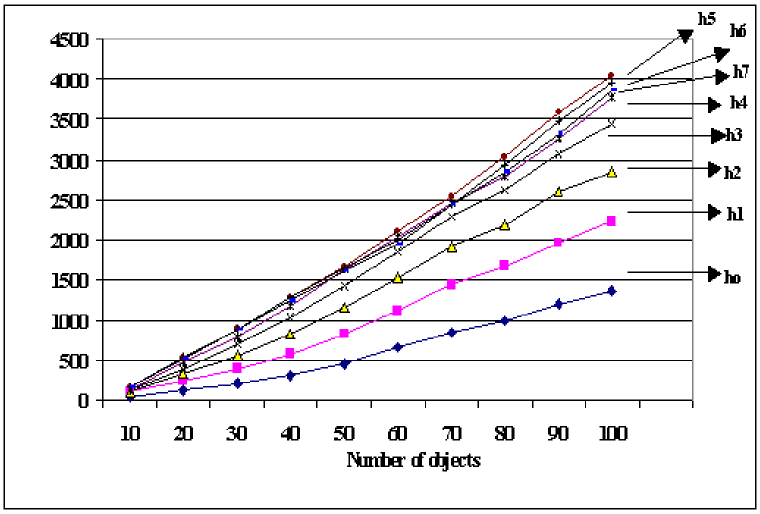

Figure 5: Number of links vs number of objects, under T.

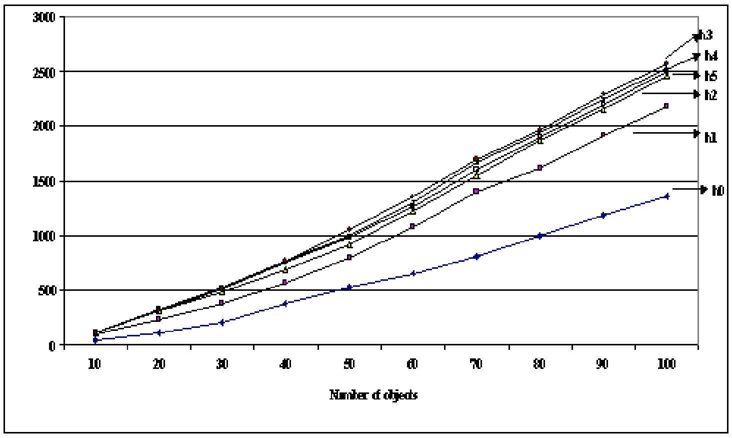

Figure 6: Number of links vs number of objects, under $T^\star$.

3.2 Number of Links

Figure 5 and Figure 6 show the growth in number of links (using T and $T^\star$ respectively). We see that the growth in the number of links is directly proportional to the growth in the number of nodes. This was expected because both have a linear growth in n, the number of objects. As a matter of fact, since the number of triplets in each G_i is $O(k)$ and since only non-empty intersection sets of triplets create classes, both the number of nodes and links are $O(nk)$ in the worst case. So the gain in the number of nodes implies an equivalent gain in the number of links, i.e., about 20%. Further reduction in the number of nodes will result in equivalent gains in the number of links, simplifying the classification structure even more. Additional pruning is thus also relevant to improving (reducing) the number of links in the resulting structure.

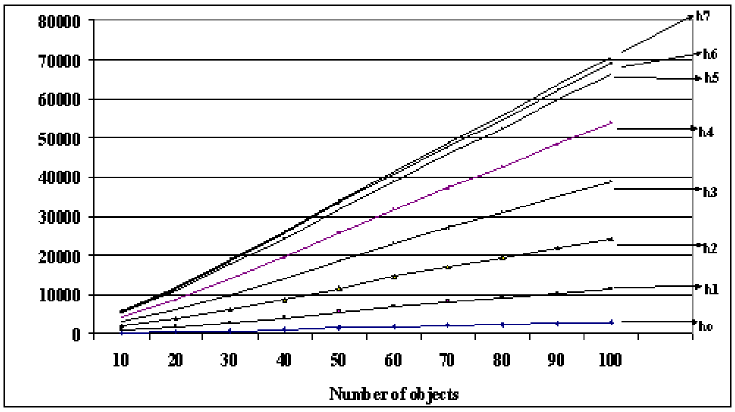

Figure 7: Number of triplets vs number of objects, under T.

3.3 Number of Triplets

As expected, the increase in the number of unifying terms resulted in an important overall increase in the size of the classification structure. As theoretically assessed in Section 2.2.1, even though the expected size of the classification structure was $O(nk)$ with a hidden constant of 16 in the worst case, the actual evaluation of this constant was around 1.5 (with the original version of the MSG). This was due to the fact that the triplets containing ? symbols were so general that they were shared by many objects, reducing the size of L_g. With the proposed pre-pruning approach, the generalizing terms are more specific than the ? symbol, and the triplets so generated are shared by fewer objects. Consequently, we observed a tremendous increase in the number of triplets. Provided that we compute h_3 using T, we end up (see Figure 7) with a classification structure which now carries about 39,000 triplets from an initial set of 3,000 triplets, which sets the hidden constant at 13. With $T^{\star}$, the hidden constant would be about 8 (see Figure 8), as a total of about 25,000 triplets is produced for h_3 thru h_5 independently. Even though the constant is much smaller with $T^{\star}$, our approach still has substantial space requirements; this is definitely its most constraining effect. However, there is no doubt that the resulting classes will be more specific than those of the classification structure produced by the original version of the MSG. As a matter of fact, we end up with 20% fewer classes while the number of triplets is increased by 3.33. So each class will have a more specific description in terms of the number of triplets composing it, and in terms of the concept types used in these descriptions (because overly general triplets are not part of the structure anymore). This proves that, after pruning, the classes carry more specific semantics than before.

3.4 Computing Time

In either case, as can be seen in Figure 9 and Figure 10, even though there is a slightly higher cost in terms of computing time, the actual platform is fast enough to handle large sets of objects. Extrapolation from either one of these two figures tells us that very large classification structures could be produced in precompilation mode in reasonable time. Consequently, computing time is not considered to be a problem.

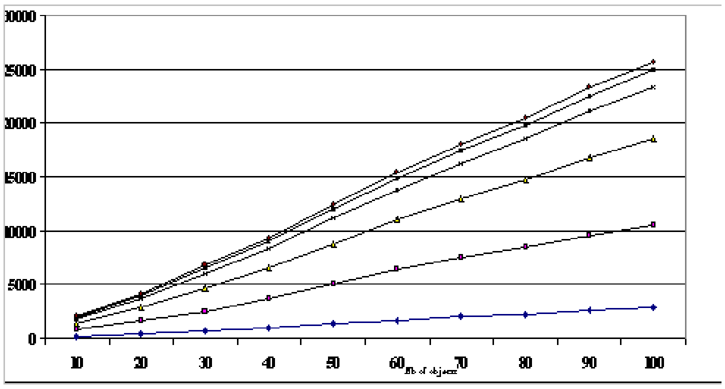

Figure 8: Number of triplets vs number of objects, under $T^{\star}$.

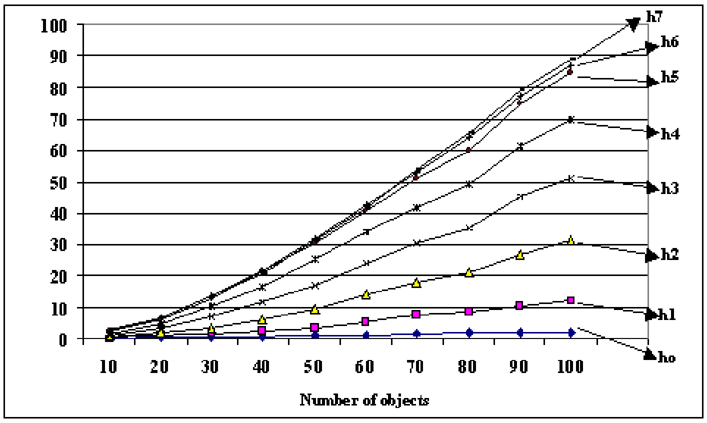

Figure 9: Computing time (in sec.) vs number of objects, under T.

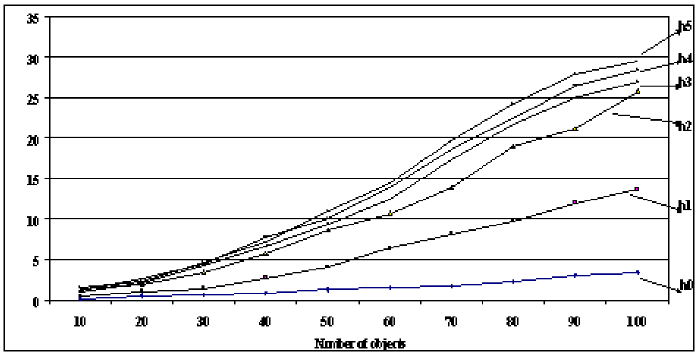

Figure 10: Computing time (in sec.) vs number of objects, under $T^{\star}$.

3.5 Summary of Comparison of MSG using T and $T^\star$

It is clear from the figures above that all performance metrics indicate that the fragmented approach, when available, should be preferred: it minimizes the added complexity associated with multiple generalizations of concept types. For instance, h_3 in the unified vocabulary is worse than any classification structure built with the fragmented vocabulary (according to *all* performance metrics). Furthermore, it was shown that the classes produced under this fragmented vocabulary approach are more specific than those produced using a unified vocabulary.

Compared with the original version of the MSG, the number of nodes and of links was cut down by 20%. However, the size of the resulting classification structure may be a problem: memory requirements are not negligible as the hidden constant went up from 1.5 to 8 times the size of the original knowledge base (due to the introduction of new triplets). Finally, the increase in computing time was not considered a problem.

The evaluation done in this section reveals that a fragmented vocabulary approach should be preferred over a unified vocabulary approach. However, as shown with our test database, the maximum length path l could have been set to any number between 3 and 5 and the gain would have been about the same (in terms of the metrics that we used). Consequently, additional pruning of the classification structure may prove to be useful, as discussed in Section 4 below.

4 Post-Pruning

4.1 Pruning Useless Links

According to Section 2.2.1, a link k exists between two classes c_1 and c_2 of C', written $c_2 < c_1$ if $k \in L(C')$. We also say that c_1 is the (immediate) *parent* of c_2 and c_2 is said to be the (immediate) *child* of c_1. In general, let us define the set of immediate children of a class p as $C(p) = \{$class c in the classification structure, for which there is a link between p to c (with $p > c)\}$. Similarly, we can define the set of the immediate parents of a class c as being $P(c) = \{$class p in the classification structure, where $c \in C(p)$ (with $p > c)\}$. Finally, let us define the set of objects represented by a class c as $E(c)$, called the *extension set* of class c.

Either in terms of an entropy metric such as found in traditional decision tree literature (Quilan, 1986), or in terms of the discrimination power of particular objects (Dietterich & Michalski, 1983), or in terms of the optimal height of a classification structure, the most useful links are those which relate a class p to a subset of its children nodes C'(p) such that $\forall c \in C'(p)$, $c \in C(p)$ and $abs((E(c)/E(p)) - 0.5) < \varepsilon$, where *absc()* is the absolute value function and *epsilon* is a small constant value. Let us define the *selectivity* of link k as $sel(k) = E(c)/E(p)$. Knowing that each time a link is replaced by some other link to a more specific class, the selectivity of the newly added link is less than the selectivity of the deleted link, we now propose a pruning algorithm that deletes links whose selectivity is strictly higher than a certain threshold[6].

[6]Deleting links whose selectivity is below some threshold is pointless; the se-

1. $M \leftarrow \emptyset, N \leftarrow \emptyset$.
2. $L \leftarrow$ [the set of all *links* in the classification structure] (or some other starter set)
3. Compute $sel(k)$ for all links k in L.
4. For each link k (between p and c, with $p > c$) such that $sel(k) > t$, a predefined threshold, the content of c is copied downwards into all its immediate children nodes, i.e., into each class of $C(c)$, so that no information is lost. Then $M \leftarrow M \cup \{c\}$ and $N \leftarrow N \cup \{p\}$ (for the next iteration)
5. For all $c \in M$ still in the structure, compute $P(c)$ and $C(c)$ and reconnect all p' in $P(c)$ to all c' in $C(c)$ unless there is already a path between p' and c' in the classification structure, and delete c and all its connecting links.
6. If $M \neq \emptyset$ then: $L \leftarrow$ {links emanating from the classes in N}, $M \leftarrow \emptyset$, $N limp\emptyset$, goto step 3.
7. Stop.

This algorithm is simple enough to be applied on large classification structure. Though we did not assess its asymptotic complexity yet, the experiments reported below show that a very small portion of the links were affected by it in our data set. When applied onto h_5 (built with a fragmented vocabulary) with a threshold of 0.65, the number of links in the [0.40..0.60] selectivity interval increased by only 2%. Furthermore, no gain in the number of nodes was achieved while all other performance metrics showed important drawbacks. This is sufficient to state that the negative side-effects of the method outweigh the slight increase in the selectivity of certain links. Therefore we can not see where such a technique could be applied successfully on large size applications where storage requirements are important.

4.2 Pruning Useless Classes

The other pruning technique that we propose in this section uses a discrimination metric, *ent*(.), based on some entropy evaluation of each class in the classification structure. It is evaluated as:

$ent(p) = (\Sigma_{c \in C(p)} abs(E(c) - E(p)/C(p)))/C(p)$

It favors equal distribution of the set of objects of a class into its immediate children classes (for reasons mentioned in Section 4.1 above). The classes for which this metric evaluates above some threshold t will be considered useless and will be deleted from the classification structure.

1. $M limpemptyset$
2. $L \leftarrow$ {all *classes* in the classification structure } (or some other starting set)
3. Compute $ent(p)$ for all classes p in L.

lectivity of the added links would be even lower than that of the deleted links.

4. For each class p such that $ent(p) > t$ a predefined threshold, the content of p is copied downwards into all its immediate children nodes, i.e., into each class of $C(p)$, $M \leftarrow M \cup P(p)$ (for the next iteration), each class p' in $P(p)$ is connected to each class c' in $C(p)$ unless there is already a path between p' and c' in the classification structure. Then p and its connecting links are deleted.
5. if $M \neq \emptyset$ then : $L \leftarrow M$, $M \leftarrow \emptyset$, goto step 3.
6. Stop.

In the experiments reported below, three different simple-to-compute thresholds were used: $\mathbf{s1} = (xm)/100$, $\mathbf{s2} = (x(m/k))/100$, and $\mathbf{s3} = x$, for $x \in [0, 100]$, where $m = E(c)$, and c is the class for which $ent(c)$ is computed and where k is the number of children of class c. All thresholds express an acceptable variability on the entropy of a class. The class will be deleted if $ent(c) > t$. The two thresholds, $s1$ and $s2$, are called *local* thresholds since $E(c)$ is part of their evaluation; while **s3** is a *global* threshold since it is fixed for all classes. Of course **s3** requires no computation, but is harder to assess. Unlike **s1** which has more impact on more specific classes, **s3** is more effective near the top of the classification structure: no classes c for which $E(c) < \mathbf{s3}$ will ever be pruned. Threshold **s2** is more drastic than the other thresholds, forcing a partition of extension sets of a class into its immediate children classes. Since our classification structure is not a tree, **s2** may overprune the structure. The precise determination of correct values for each of these parameters can be determined based on experience. The following sections present our findings with that regard, and compare the three approaches (thresholds) that were used.

4.2.1 Number of Nodes

Figure 11 shows how the choice of the threshold affects the elimination of nodes in the classification structure. As expected, **s2** produced a drastic pruning right from the start: this may introduce important degeneration of the classification structure. However, in what follows, in order to fully compare the different thresholds, the next subsections evaluate some performance metrics on classification structures having the same number of nodes after having been pruned using **s1**, **s2** or **s3** as threshold.

4.2.2 Redundancy: the Number of Links and Additional Triplets

Figure 12 gives a comparison basis for classification structure h_5 (built using a fragmented vocabulary) and pruned using the different thresholds. For the three thresholds, the number of links decreases gradually. The more drastic gain in the number of links is obtained with **s2**. However, Figure 13 shows the added complexity in terms of triplet redundancy, and **s2** is identified as the worse case with regard to that parameter which was earlier identified as a potential problem with large size applications (see Section 3.5). In this case, **s1** or **s3** could be used as threshold.

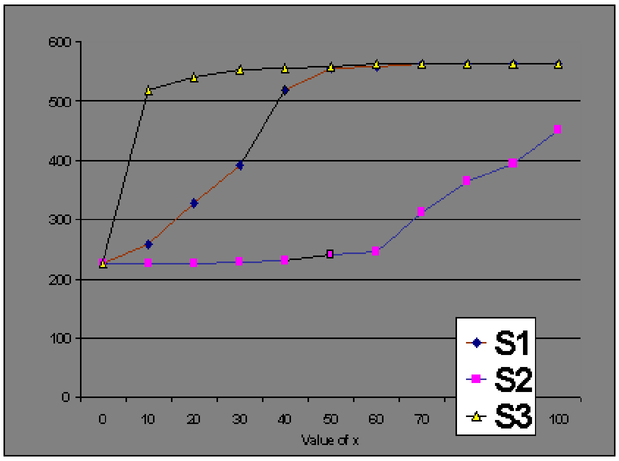

Figure 11: Number of nodes vs x.

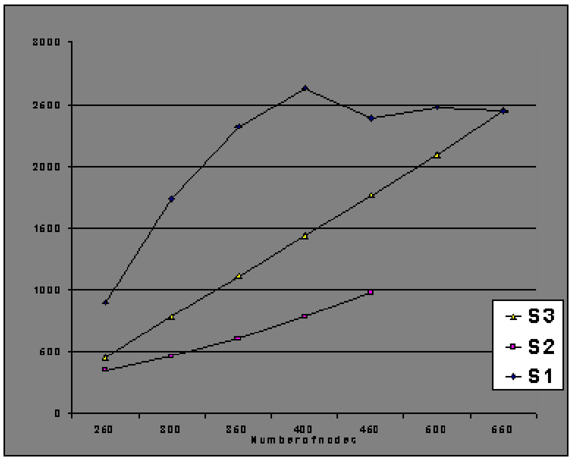

Figure 12: Number of links vs number of nodes.

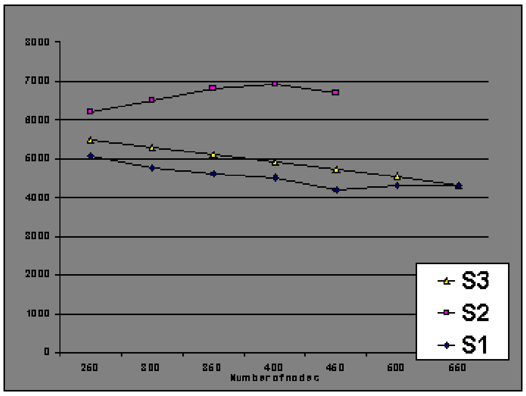

Figure 13: Number of triplets vs number of nodes.

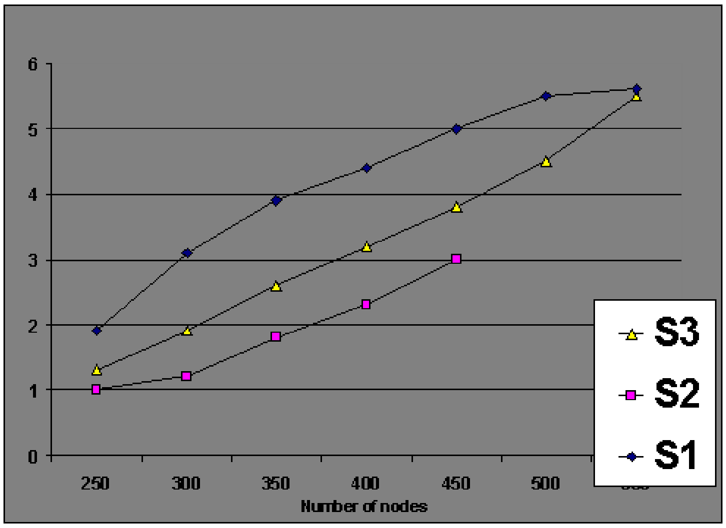

Figure 14: Average height vs number of nodes.

4.2.3 The Height of the Pruned Classification Structure

As shown in Figure 14 below, the gain in average height is not significant. Once again, **s2** is identified as potentially generating too much pruning: a classification structure with not enough height is a typical sign of a degenerate structure (which would not characterize the domain well). Here also, **s1** and **s3** seem to have a similar effect on the structure, but on different nodes. However, like with the number of links, **s3** slightly outperforms **s1**.

5 Conclusion

In summary, this paper presented simple pre and post-pruning techniques for large classification structures. The use of a fragmented vocabulary in the generalization process of objects clearly improves the standard performance metrics used to assess the quality of such structures. Among other things, the number of nodes and links was reduced by at least 20%. The only drawback is memory requirements which may be a serious problem. In any case, it is clear that class descriptions are more specific than with either the original version of the MSG or its unified vocabulary version. It was also clear that additional pruning could reduce even more the size of the resulting classification structure. Therefore, this paper also presented post-pruning techniques.

The first post-pruning technique aimed at deleting weak links in terms of their discrimination power. This experiment failed under our data set, and seemed to indicate that links are rather incidental with regard to the characterization of a domain, and that therefore, there is not much point in trying to change the topology of a classification structure in that way. Consequently we considered post-pruning methods based on a simple-to-compute entropy metric on classes. Three different thresholds, allowing different variability on the entropy of a class, were used: **s1** (variability with regard to the extension set of the parent class), **s2** (variability with regard to the ideal partition of the extension set of the parent class) and **s3** (variability expressed in terms of a number of objects). Threshold **s2** proved to over-prune the structure. Threshold **s1** adapts to the class for which it is used: it is a local threshold; while threshold **s3** is fixed for all classes. Based on a percentage of the extension set of the class, **s1** has more effect on the specialized classes; while **s3** has more effect on general classes. Threshold **s3** performs better than **s1** in all cases, except for a slight increase in the number of triplets. In cases where space requirements become a major constraint (see Section 3), it may be better to choose **s1** rather than **s3**. However, depending on the availability of background knowledge, one may be harder to assess than the other. In any case, both techniques proved to be useful.

Future developments involve the precise assessment of the asymptotic complexity of these methods and their evaluation on other application domains. Despite these upcoming results, it is clear that the methods proposed in this paper are simple to implement and to apply, and it is foreseen that their complexity is manageable. This remains to be fully investigated and will be the subject of a forthcoming paper.

6 References

Bournaud, I., (1996). Regroupement conceptuel pour l'organisation de connaissances. Ph.D. Thesis, Université Paris VI – Institut Blaise Pascal, France.

Dietterich, T.G. & Michalski, R.S., (1983). A Comparative Review of Selected Methods for Learning from Examples. In: Machine Learning: An Artificial Intelligence Approach. R.S. Michalski, J.G. Carbonell & T.M. Mitchell (Eds.). Morgan Kaufmann. 83-129.

Godin, R., Mineau, G.W. & Missaoui, R., (1995a). Incremental structuring of knowledge bases. In: Proceedings of the 1st International Symposium on Knowledge Retrieval, Use, and Storage for Efficiency (KRUSE-95). Santa Cruz, California. Août. G. Ellis, R.A. Levinson, A. Fall & V. Dahl (Eds.). Department of Computer Science, University of California at Santa Cruz. 179-193.

Godin, R., Mineau, G.W., Missaoui, R. & Mili, H., (1995b). Méthodes de classification conceptuelle basées sur les treillis de Galois et applications. In: Revue d'Intelligence Artificielle, volume 9, no 2. 105-137.

Michalski, R.S., (1983). A Theory and Methodology of Inductive Learning. In: Machine Learning: An Artificial Intelligence Approach. R.S. Michalski, J.G. Carbonell & T.M. Mitchell (Eds.). Morgan Kaufmann. 41-81.

Mineau, G.W., (2000). A Universal Representation Paradigm for Knowledge Base Structuring Methods. Academic Press International Theme Volumes on Knowledge Based Systems, Techniques and Applications. C.T. Leondes (Ed.). Academic Press.

Mineau, G.W. & Godin, R., (1995). Automatic Structuring of Knowledge Bases by Conceptual Clustering. In: IEEE Transactions on Knowledge and Data Engineering, volume 7, no 5. 824-829.

Quinlan, J.R., (1986). Induction of Decision Trees. In: Machine Learning Journal, volume 1. Kluwer Academic. 81-106.

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

Wille, R., (1982). Restructuring Lattice Theory: an Approach Based on Hierarchies of Concepts. In I.Rival (Eds.), *Ordered Sets,* (pp. 445-470). Dordrecht-Boston: Reidel.