Automatic Structuring of Knowledge Bases by Conceptual Clustering

(Final version appeared in *IEEE Transactions on Knowledge and Data Engineering*, 1995, **7**(5), 824-828.)

Guy William MINEAU[1], **Robert GODIN**[2]

ABSTRACT[3]

An important structuring mechanism for knowledge bases is building an inheritance hierarchy of classes based on the content of their knowledge objects. This hierarchy facilitates group-related processing tasks such as answering set queries, discriminating between objects, finding similarities among objects, etc. Building this hierarchy is a difficult task for the knowledge engineer. Conceptual clustering may be used to automate or assist the engineer in the creation of such a classification structure. This article introduces a new conceptual clustering method which addresses the problem of clustering large amounts of structured objects. The conditions under which the method is applicable are discussed.

indexed terms: conceptual clustering, data structures, knowledge bases, knowledge indexing, machine learning

1. Introduction

A classification structure is a set of hierarchically related classes, where hierarchically higher classes subsume their lower counter-parts. It can be viewed as an index. Its utility is obvious from an efficiency point of view. Building such a structure is a difficult task for the knowledge engineer. Conceptual clustering methods may be used to automate or assist the engineer in this process [1]. The conceptual clustering method that we propose has attractive distinguishing characteristics with respect to other methods. The next section describes the better known methods and highlights the major differences of our proposal.

2. Background

The method we propose has three main objectives: it has to be applicable on structured objects (graph-described objects), on large-sized knowledge bases, and it has to be incremental (as knowledge bases may require frequent updates and revision). The inherent combinatorial complexity of classifying arbitrary graphs necessitates the introduction of simplifying assumptions and the use of heuristics in order to reach a certain degree of feasibility. Consequently, we highlight the assumptions each method makes in order to cope with the complexity issue.

A large number of methods has been considered in the litterature. From them, less recent methods such as EPAM [2], COBWEB [3], UNIMEM [4], CLUSTER/2 [5], and concept analysis [6] are limited to unstructured descriptions such as

attribute-value pairs, and are therefore inadequate for richer knowledge representations such as needed with structured objects. More recent methods have been proposed to address richer representations.

The CLUSTER/S system [7] was designed to handle structural descriptions based on a typed predicate calculus description language. CLUSTER/S pre-analyzes the objects it has to process, transforming their structural description into attribute-value pairs, introducing the important clustering-related attributes. Then CLUSTER/2 can be used. In order to proceed as such, the user must know in advance which attributes are clustering-relevant, and which are not. Background knowledge on the clustering criterias is essential. Generally, such conditions are not satisfied.

Lebowitz's RESEARCHER [4] system claims to deal with complex structural descriptions, but very little details are given. All the examples found in the litterature are based on structured objects which share a strong structural similarity (as with frame systems). Lebowitz does point to the difficulty of the matching process with complex objects. However no claim of handling large amounts of knowledge objects could be made until a detailed complexity analysis is available, or until extensive experimentation with real data is accomplished.

The CLASSIT system [8] improves on COBWEB by dealing with numeric attributes and by allowing richer descriptions of instances (defined as sets of elements). However, every element in a set must have the same structure which is used to simplify the matching process. Thus, the representational capabilities of this system are too limited for our purposes.

Thompson and Langley's LABYRINTH system is also based on COBWEB [9]. It addresses the problem of classifying composite objects, i.e., objects having other objects as parts. Because a description of an object follows a tree-like pattern, COBWEB can be applied recursively on all the different parts composing the object. Eventhough this attempt is one of the closest ones to our goal of applying conceptual clustering methods to structured objects, representational expressiveness is still limited, though improved over the preceding methods.

Formal concept analysis is introduced in [10]. It is based on generating the Galois lattice of a binary relation R between a set of instances I and a set of features F. A Galois lattice can be considered as a concept hierarchy where each node of the lattice corresponds to a concept. The concept is defined by a pair (X,X') where X, the extension of the concept, is a subset of I, and X', the intension, is a subset of F, both satisfying the following: 1) X' is the set of features common to all the instances of X, 2) X is the set of instances having at least the features in X'. The resulting structure does not depend on any particular order of introduction of new instances. Another important distinguishing feature is that the hierarchy is not a tree as in the preceding methods (useful for multiple inheritance systems).

The work presented in this paper may be viewed as an extention and refinement of concept analysis for dealing with a richer representation language. The instances are represented by conceptual graphs[4] [11], which is a well-defined and

established knowledge representation formalism. The hierarchy generated, called a *knowledge space* (KS), is not necessarily a tree and not necessarily a lattice (as opposed to concept analysis). This difference is due to the fact that only a subset of the Galois lattice is generated to cope with the added complexity of dealing with a richer description language.

Section 3 below introduces our approach and illustrates the principles of the method on a small example, gives a formal definition of a KS and presents and analyzes the algorithms which build the KS. Section 4 gives some experimental results. Section 5 discusses the shortcomings of the method.

3. Conceptual Clustering of Structured Objects: our Solution

Automatic classification of graphs introduces complexity problems related to the graph matching headache. Hence, we propose a method that will classify graphs using their *common subgraphs*. Classes will be formed according to these extracted common subgraphs. The derivation of complete characteristic descriptions for these classes will be postponed to a later stage, only when the application will need them. This is done to alleviate the classification process which can be restricted to the detection of common subgraphs, a much simpler problem. To be effective, this method is based on the following hypothesis: *similar semantics will be encoded using similar syntactical forms*. When that hypothesis holds, the graphs are said to share *structural similarity*. The problems associated with this heuristic are discussed in section 5, and ways of improving it are presented in [12,13].

Subsection 3.1 presents a simple example aiming at giving an intuitive idea of the method and of the resulting classification structure called *knowledge space*. Subsection 3.2 formally defines a knowledge space. Subsection 3.3 sketches the algorithms needed to construct the knowledge space.

3.1. A Small Illustrative Example

Our knowledge objects are described using the conceptual graph formalism of Sowa [11]. Conceptual graphs represent basic concepts using labeled boxes; relations between these concepts are expressed by labeled circles. Arcs exist only between concept and relation nodes. For example, let us consider three graphs, each describing the visual content of different documents in an image database (see Figure 1)[5]. These graphs constitute knowledge on the documents, and are called *knowledge objects* (KO). Furthermore, the concepts appearing in a graph are hierarchically related to other concepts, in a class/subclass hierarchy, called *type hierarchy* [11]. Part of our type hierarchy is represented in Figure 2. From the three graphs of Figure 1, a hierarchy corresponding to the full power set of the set of objects is illustrated in Figure 3. The contents

of the nodes of this structure represent the characteristic descriptions for the graphs describing the class. To infer these descriptions, Figure 2 was used so that non-matching concepts could be replaced by their common minimal supertypes.

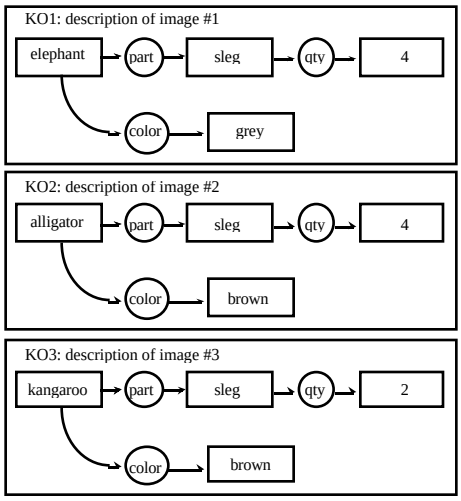

Figure 1: The description of three visual documents using conceptual graphs.

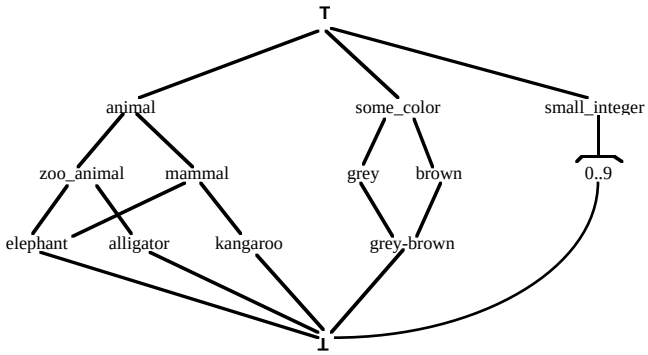

Figure 2: The type hierarchy for our small example.

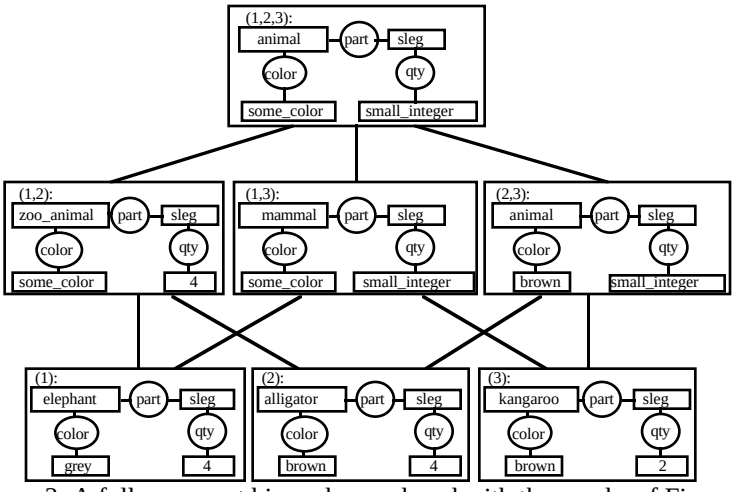

Figure 3: A full power set hierarchy produced with the graphs of Figure 1.

A knowledge space is a specialization of a full power set under the following three restrictions. First, only identical concepts and relations in all graphs forming a class will be kept in the corresponding node of the description of the class.

Those that are not identical will be replaced by a wildcard value "?" (R1). Secondly, any subgraph will appear only in the most general class containing it, producing an inheritance hierarchy (R2). After this, all nodes having an empty description will be deleted from the hierarchy (R3). Figure 4 shows the content of node (2,3) where "?" have been introduced because the concept types "alligator" and "kangaroo" did not match, and neither did "4" and "2". Figure 5 shows the content of node (2,3) after R2 was applied. Part of the description was deleted because it already appears in node (1,2,3). Figure 6 shows the corresponding knowledge space (including some information needed for the generation of characteristic descriptions).

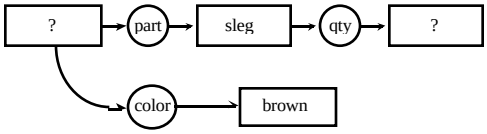

Figure 4: The content of node (2,3) after R1.

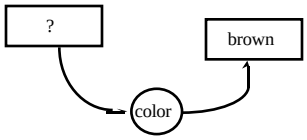

Figure 5: The content of node (2,3) after R1 and R2.

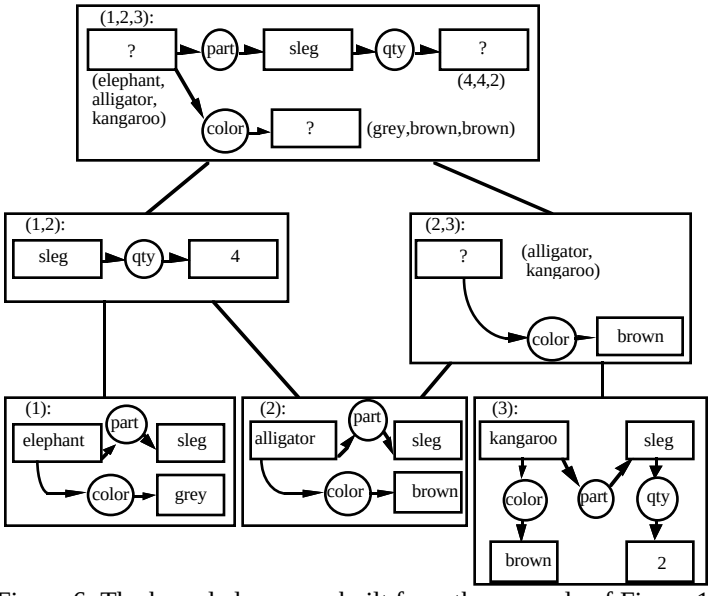

Figure 6: The knowledge space built from the example of Figure 1.

3.2. Formal Definitions

A knowledge space is a data structure which we define more formally in this subsection. Let O be the set of objects to be described; let D be the set of all descriptors in the description language. The description of all objects can be represented as a relation between D and O. Let us call this relation R, and let us define it over D x O (i.e., $R \subseteq D \times O$). Naturally, not all objects may be described at a certain point in time, and not all descriptors may be used. Let $OB(R) = \{o \mid o \in O \text{ and } \exists d \text{ such}$

that $d \in D$ and $<d,o> \in R\}$. In other words, OB(R) is the set of objects of O described by elements of D through the relation R. Similarly, we can state that $DE(R) = \{d \mid d \in D$ and $\exists o$ such that $o \in O$ and $<d,o> \in R\}$. DE(R) is the set of descriptors used to describe objects of O through the relation R. $OB(R) \subseteq O$ and $DE(R) \subseteq D$. Since not all objects are described using all the descriptors, R is a subrelation of DE(R) x OB(R), i.e., $R \subseteq DE(R) \times OB(R) \subseteq D \times O$.

Furthermore, letting o belong to O, we can define $ds(o) = \{d \mid d \in D$ and $<d,o> \in R\}$. The function ds(o) is the *descriptor set* of o. Similarly, letting d belong to D, we can define $os(d) = \{o \mid o \in O$ and $<d,o> \in R\}$. The function os(d) is called the *object set* of d. It is obvious that $ds(o) \subseteq DE(R) \subseteq D$, and that $os(d) \subseteq OB(R) \subseteq O$, $\forall o \in O$ and $\forall d \in D$.

Of course, similar objects will be described using sets of descriptors whose intersection will not be empty. This will be the criteria chosen to detect similarity among objects and to form clusters. However, the method we propose will form a cluster with the *largest* intersection set of the descriptors describing the objects of the cluster. This is done in order to keep memory and processing requirements as low as possible. The formal definition of a cluster is given below.

First, let us define a *group*. A subset G of R is a group iff $\forall d,d' \in DE(G)$ then os(d) = os(d'). As a corollary of that, if G_1 and G_2 are two groups, if $G_1 \neq G_2$ and if $OB(G_1) = OB(G_2)$, then $\exists G_3$, a group, such that $G_3 = G_1 \cup G_2$. G_3 is said to be an *extension* of G_1 and of G_2, that is, $G_1 \subset G_3$ and $G_2 \subset G_3$. Since G_3 is a group, $G_3 \subseteq R$, and more generally, $G \subseteq G_3 \subseteq R$, where $G = G_1$ or $G = G_2$.

Some groups may not be extended. They are called *maximally extended groups*. When it is impossible to find G_2 in order to extend G_1 such that $G_1 \neq G_2$, then G_1 is said to be maximally extended. If G_1 and G_2 are two different maximally extended groups, then $G_1 \cap G_2 = \varnothing$. Letting G* be a maximally extended group associated with G, then, $G \subseteq G^* \subseteq R$.

Consequently, every tuple of the relation R, <d,o>, belongs to only one maximally extended group. In effect, since every maximally extended group is pairwise disjoint of any other maximally extended group, a tuple must belong to a maximum of only one of such groups. Further, it must belong to some group. In effect, let G be a maximally extended group. A tuple, <d,o>, will belong to G iff OB(G) = os(d). If no existing maximally extended group satisfying this condition exists, then {<d,o>} will be de facto a maximally extended group.

Because each tuple of R belongs to exactly one maximally extended group, R can be decomposed into a partition P of such groups. A knowledge space (KS) is a classification structure which represents this partionning P. Each node of a KS represents a different maximally extended group of R. Let G be such a group. There will be a node in the KS representing G. This node will represent the cluster identified by $OB(G)^6$, and will contain DE(G).

All nodes of a KS are hierarchically related through the use of links. These links represent the partial order of inclusion defined over the sets of objects that each cluster represents. That is, let N_1, N_2 and N_3 be three nodes representing

respectively three maximally extended groups G_1, G_2 and G_3 in P. N_1 will be connected downward to N_2, written down_link(N_1,N_2) and up_link(N_2,N_1), iff $OB(G_2) \subset OB(G_1)$ and there is no other node N_3 representing G_3 in P such that $OB(G_2) \subset OB(G_3) \subset OB(G_1)$.

3.3. The Algorithms

This subsection gives implementation details on how a knowledge space can be constructed. It outlines the algorithms used to create a KS from knowledge objects, and presents the data structures needed to reach that goal. The algorithms are characterized by worst-case complexity analyses of their memory and processing requirements.

3.3.1. Constructing R

Using binary relations only, all knowledge objects can be described using triplets of the form <concept$_1$,relation,concept$_2$>, where relation is a directed binary relation between concept$_1$ and concept$_2$. Each triplet will be generalized using the "?" symbol. That is, each component of each triplet will be replaced alternatively by the "?" symbol. Consequently, the generalization of a triplet will produce 8 generalized triplets called *generalization patterns*, including the original triplet. Under the assumption that each object is described by a bounded number of triplets, kmax, the generalization process produces a descriptor set for this object whose cardinality is bounded by 8 kmax. All generalization patterns generated from all descriptor sets constitute D. The cardinality of D is then bounded by 8 kmax n, i.e., is O(n), where n is the cardinality of O.

R will be built incrementally while scanning the objects in O. In the implementation, R is called *intersection matrix* (IM), and will be internally represented by a sequential list of *associations* between a descriptor and a set of knowledge object identifiers. For instance, if descriptor d_1 is found in the descriptor sets of O_1 and O_2, the following association will be part of the sequential list describing R: <d_1,{O_1,O_2}>. In fact, the asssociation will be: <d_1,os(d_1)>, where os(d_1) only takes into account the objects scanned so far. For simplicity, the sequential list structure was first adopted. An indexed data structure could improve significantly the performance of our algorithms. Here is the algorithm that builds R.

For all o in O **do**: (* the number of objects in O is n *)

  For all d in ds(o) **do**: (* the number of descriptors in ds(o) is bounded by 8 kmax *)

    Update <d,os(d)> or Add <d,{o}> to R if absent.

The innermost instruction is done O(n) times. Each time it is executed, a sequential search of R is done over the total number of descriptors, which is O(n), the cardinality of D. Adding a new association or updating an old one is done in O(1). The processing time for this first algorithm is thus $O(n^2)$. However, because general descriptors, those having one or two "?" symbols as components, will appear in different descriptor sets, the overlap that this creates will help the search for the association <d,os(d)>. In effect, this search, though sequential, will stop earlier in the association list since general descriptors may have been already introduced into the list by the scanning of previous objects. This will reduce the complexity associated with the search of R. The space requirements for R are O(n): it is proportional to each addition to R.

3.3.2. Finding the Nodes of the KS

As stated in subsection 3.2, each node of a KS represents a maximally extended group of R. Each of these groups will be computed incrementally as we scan R. In effect, we know that each element of R belongs to only one maximally extended group G, i.e., to only one node. We also know that this node is identified by OB(G), which is identical to os(d) for $\forall d \in DE(G)$ by definition. Thus we only have to add d to DE(G). Here is the algorithm that produces the nodes of the KS.

For all d in DE(R) **do**: (* sequential scan of the intersection matrix *)
    Find the node N labeled by os(d); (* using a tri where all labels are incrementally inserted *)
    If N does not exist **then**: create it; **Set** DE(G) **to** $\varnothing$;
    Construct $DE(G) = DE(G) \cup \{d\}$.

Since each object can only be described by 8 kmax descriptors, the identifier of one object can only be found in 8 kmax object sets, i.e., in 8 kmax labels, and thus can only appear in 8 kmax associations. So, the summation of the cardinality of the right side of all associations <d,os(d)> is O(n). That is, $\Sigma_{d \in DE(R)} os(d) \leq 8\ kmax\ n$. Finding a node N requires that a tri be searched for each os(d), the label of the node representing the maximally extended group to which d belongs to. Consequently, finding each node associated with each d in DE(R) can be done in O(n) steps. The size of the tri is also bounded by 8 kmax n for the same reasons. The last instruction is executed only O(n) times, the cardinality of DE(R), so the total processing time for this algorithm is O(n). Since each descriptor is transferred to one node of the KS, the total size of these nodes is $O(n)^7$.

3.3.3. Linking the Nodes of the KS

Nodes will be inserted into the KS one by one; the KS will be constructed incrementally. A breath-first search of the tri will help scan the nodes in increasing order of their label size[8]. This scan will permit to consider the insertion of a new node only in relation to its potential children. This simplifies the task.

In order to find the children of a node N, the label of N will be scanned. Let G be the maximally extended group represented by N. Let L = OB(G), the label of N. The children of N will be found by searching through the KS starting with the nodes whose label is exactly one component of L. These nodes are called *base nodes* of L, and represent potential children of N. They must not be connected to N if one of their parents is a potential child of N, as seen in subsection 3.2. Starting with these base nodes, we will look for parents of these nodes which could be children of N. If we do not find such parent nodes, then we can link the corresponding base node to N, otherwise we start recursively with the parent of a base node.

For all node N in the tri **do**: (* sequential breath-first scan of the tri, a O(n) process *)

For all o in L **do**: (* where L is the label of N *)

Find the base node whose label is {o}, called N_b; (* done in O(1) time, special case *)

Connect(N_b,N).

Procedure Connect(N_0,N):

Set indirect_link **to** FALSE;

For all node N_i such that up_link(N_0,N_i) exists **do**: (* $i \leq 8$ kmax *)

If $L_i \subset L$ **then:** (* where L_i is the label of N_i *) **Set** indirect_link **to** TRUE;

Connect(N_i,N);

If not indirect_link **then**:

If up_link(N_0,N) does not exist **then**: create it;

create down_link(N,N_0).

Connect(N_0,N) is a recursive procedure which will connect directly two nodes iff there should not be an indirect link between them. In any case, when Connect finishes, there is a bidirectional path between N_0 and N.

This algorithm takes $O(n^2)$ steps in the worst case. Since any object can not be part of more than 8 kmax labels, scanning every node's label is a O(n) process altogether. Consequently, Connect(N_b,N) is executed a total of O(n) times. Each execution of Connect(N_b,N) is a search for potential children of N, starting at node N_b. Even if all ancestor nodes of N_b are visited, the number of visited nodes is O(1) with respect to n, the number of objects. At each visit though, a label inclusion test needs to be done ($L_i \subset L$). In the worst case, since the labels are kept sorted, inclusion may be determined by a O(n)

process. Consequently, each call to Connect(N_b,N) may take O(n) steps in the worst case. Since Connect(N_b,N) is called O(n) times, this third algorithm is O(n^2) in the worst case.

The additionnal size of the KS for storing the links between nodes is O(n), giving a total space requirement of O(n). Since the number of parent nodes of one node is bounded by 8 kmax, the total number of uplinks is O(n). Because, there is one downlink for every uplink, the upper bound on space requirements remains O(n).

Section 4 below gives empirical results about time and memory requirements as evaluated by the implementation of the method on a particular application, and compares these findings with the complexity analyses of this present section.

4. Implementation

This section describes a prototype implementation of the method and an experimental application on an image database. The implementation was realized in C on an Amiga 2500 platform. It is based on the formal model of section 3.2 and on the algorithms described in section 3.3.

4.1. Test Database

The database used is a collection of 100 images of artifacts from the National Gallery of Arts of Washington. Each image is described by a conceptual graph. One part of the graph encodes a description of the subject itself and another part encodes bibliographic information related to the artwork, such as the name of the author, date of the work, materials used, etc. The total conceptual vocabulary is composed of 400 concepts, 80 relations and 300 individuals. Individuals are instantiated concepts represented by proper nouns. The concepts and relations are organized in two separate taxonomic hierarchies.

Graphs were first decomposed into a set of triplets. In our test database, the mean number of triplets per object is only 29.82, with a standard deviation of 8.79. The largest graph has 53 triplets, and the smallest, 16 triplets. This was very surprising and added faith to the restriction that the size of each graph should be bounded by a constant.

4.2. The Results

Figure 7 shows the evolution of space requirements observed versus the number of objects. For 100 objects, we needed 494,070 bytes. The data is consistent with the linear worst-case analysis of section 3. The resulting KS had 571 nodes with an evolution proportional to n, the number of objects.

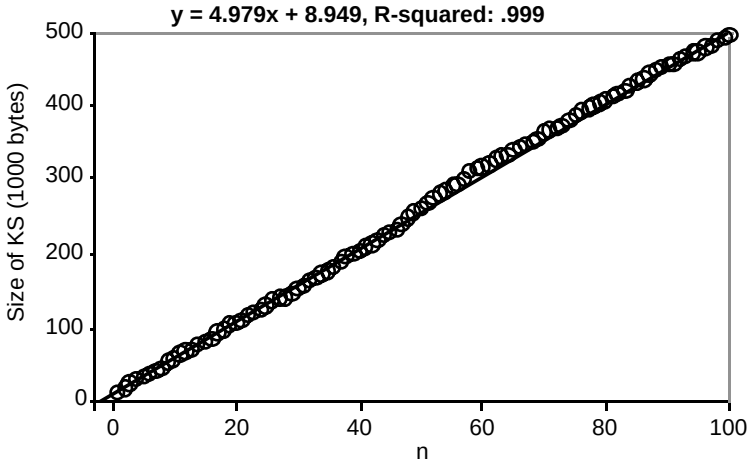

Figure 7: Evolution of the size of both the KS and matrix in bytes vs n, the number of objects.

The total processing time was 13.6 minutes, the major part of which was needed to build the matrix (11.54 minutes). The major complexity factor is the lookup of a particular GP entry in the matrix (sequential search for simplicity reasons). The observed time grows roughly as $O(n^{1.29})$, which is less than the quadratic worst-case analysis. This was predictable because of the overlapping between GPs which speeds up the matrix lookup process. We could significantly improve our implementation by using an index on the matrix. Observed data reveals a growth proportional to n for node generation, and to n^2 for the linking process, as expected. Figure 8 shows the evolution of the total processing time, which is sub-quadratic, i.e., proportional to $O(n^{1.36})$, because our main factor depends on the building of the intersection matrix which is $O(n^{1.29})$.

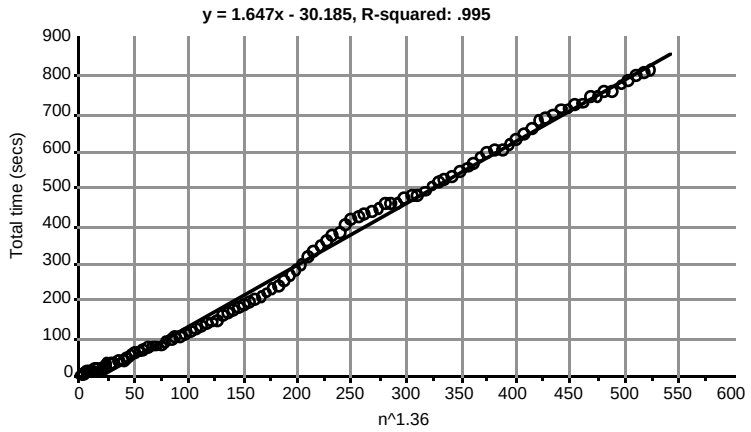

Figure 8: Total processing time for the creation of a KS vs $n^{1.36}$, where n is the number of objects.

5. Main Shortcomings of the Method

As mentioned before, the proposed method is well-adapted for applications where partial class descriptions are sufficient for characterizing the class. As this partial information is based on common subgraphs, applications which will be candidate for the method are those whose knowledge objects show some structural similarity[9]. This structural similarity, though

improvable through controlled acquisition environments [12,13], is highly dependent upon the nature of the application domain itself. So, the proposed method is not universal, but when applicable, it can be used on large-sized knowledge bases.

Also, the generalization phase of the method generalizes each object independently of each other. This creates overgeneralizations which may result in the production of more nodes in the knowledge space than useful. In effect, some nodes will represent common subgraphs from which no useful class description may be inferred since they represent either overgeneralizations or syntactical similarities which do not correspond to any semantical similarity. These classes are *useless*.

6. Conclusion

A conceptual clustering approach was proposed in order to organize the content of a knowledge base. This method aimed at providing such an organization under the following conditions: when knowledge objects contain descriptive and factual knowledge, when these objects encode knowledge with a rich formalism allowing the representation of structured objects, when a large number of objects is to be considered, when the application domain implies dynamic evolution of the knowledge base. The classification structure resulting from the application of our method, called knowledge space, contains classes of objects which share common subgraphs. When these objects encode a similar semantics using similar syntactical structures, they are said to share structural similarity. Based on this structural similarity criteria, common subgraphs are extracted and represent the content of the classes. The description of a class can be inferred from these common subgraphs used as anchor points for the matching process.

Extensive experimentations were conducted on a sample image database of 100 documents described by conceptual graphs (each of average size of 30 relations). Extrapolation from these experimentations showed that the method is realistically applicable on large-sized knowledge bases.

7. References

[1] G. Mineau, J. Gecsei, and R. Godin, "Improving Consistency within Knowledge Bases", *Knowledge, Data and Computer-Assisted Decisions*, M. Schader & W. Gaul, Eds. Springer -Verlag, 49-65, 1989.

[2] E.A. Feigenbaum, and H. Simon, "EPAM-like Models of Recognition and Learning", *Cognitive Science*, 8. 1984.

[3] D. Fisher, "Knowledge Acquisition via Incremental Conceptual Clustering", *Machine Learning*, 2, 139-172, 1987.

[4] M. Lebowitz, "Concept Learning in a Rich Input Domain: Generalization Based Memory", R. Michalski, J. Carbonell, and T. Mitchell, Eds., *Machine Learning: An A.I. Approach*, Morgan Kaufmann, 193-214, 1986.

[5] R.S. Michalski, R. Stepp, and E. Diday, "A Recent Advance in Data Analysis: Clustering Objects into Classes Characterized by Conjunctive Concepts", Kanal and Rosenfeld, *Progress in Pattern Recognition*, North-Holland, 1981.

[6] R. Wille, "Knowledge Acquisition by Methods of Formal Concept Analysis", E. Diday, Ed., *Data Analysis, Learning Symbolic and Numeric Knowledge*, New York: Nova Science Pub., 365-380, 1989.

[7] R.E. Stepp, and R.S. Michalski, "Conceptual Clustering: Inventing Goal-Oriented Classifications of Structured Objects", R. Michalski, J. Carbonell, and T. Mitchell, Eds., *Machine Learning: An A.I. Approach*, Kaufmann, 1986.

[8] J.H. Gennari, P. Langley, and D. Fisher, "Models of Incremental Concept Formation", J. Carbonell, Ed., *Machine Learning: Paradigms and Methods*, Amsterdam, The Netherlands: MIT Press, 11-62, 1990.

[9] K. Thompson, and P. Langley, "Incremental Concept Formation with Composite Objects". *Proceedings of the 6th Int.Workshop on Machine Learning*, Cornell University. Morgan Kaufmann, 1989.

[10] R. Wille, "Restructuring Lattice Theory: an Approach Based on Hierarchies of Concepts". I. Rival, Ed., *Ordered Sets*, Dordrecht-Boston: Reidel, 445-470, 1982.

[11] J.F. Sowa, "*Conceptual Structures: Information Processing in Mind and Machine*", Addison-Wesley, 1984.

[12] G.W. Mineau, "Normalizing Conceptual Graphs. *Proceedings of the Fifth Annual Workshop on Conceptual Structures*, AAAI-90, P. Eklund and L. Gerholz, Eds., Boston, 1990.

[13] G.W. Mineau, "Acquisition d'objets structurés destinés à la classification symbolique", *Proceedings of the seventh Journées Francophones sur l'Apprentissage et l'Explicitation des Connaissances (JFA-92)*, Dourdan, France, 1992

Footnotes:

1. Département d'Informatique, Faculté des Sciences et de Génie, Université Laval, Ste-Foy, Québec, Canada, G1K 7P4. Tel: (418) 656-5189, e-mail: mineau@ift.ulaval.ca.
2. Département de Mathématiques et d'Informatique, Université du Québec à Montréal, C.P.8888, Succ.A, Montréal, Québec, Canada, H3C 3P8. Tel: (514) 987-3088, e-mail: godin@info.uqam.ca.
3. This work is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), under grants #OPG0105365, #OGP0041899 and #OGP0009184.
4. In fact, they are represented by first-order conceptual graphs, i.e., those having a mapping to a first-order logic formula.
5. In this example, the concept type "sleg" stands for "supporting_leg", that is, legs allowing mobility for a particular legged animal.
6. In fact, OB(G) will become the label of the node.
7. In fact, only specific descriptors will be transferred to the KS, those carrying new information. Consequently, the size of the KS, though O(n) since DE(R) is O(n), will be much lower than the cardinality of DE(R).
8. Since the tri is O(n) big, its scan will take O(n) steps.
9. Structural similarity among knowledge objects is said to be present when objects encoding similar semantics use similar syntactical structures.