Applying Concept Formation Methods to Software Reuse

Final version appeared in *International Journal of Software Engineering and Knowledge Engineering*, **5**(1), 119-142.

Robert Godin[1], Guy Mineau[2], Rokia Missaoui[1],
Marc St-Germain[1], Najib Faraj[1]

1 - Département d'Informatique
Université du Québec à Montréal
C.P.8888, Succursale Centre Ville
Montréal, Québec
Canada, H3C 3P8
(514) 987-3088
Fax: (514) 987-8477
godin.robert@uqam.ca

2- Département d'Informatique
Faculté des Sciences et de Génie
Université Laval
Québec, Québec
Canada, G1K 7P4
(418) 656-5189
Fax: (418) 656-2324
mineau@ift.ulaval.ca

Abstract

This paper describes an approach to software reuse that involves generating and retrieving abstractions from existing software systems using concept formation methods. The potential of the approach is illustrated through two important activities of the reuse process. First, the concept hierarchy generated by the concept formation methods is used for organizing and retrieving the artifacts inside a repository. Second, the generated concepts are used in identifying new abstractions that may be converted into new more generic artifacts with better reuse potential. These experiments are part of a major software engineering research project involving many business and academic partners.

Keywords: software reuse, incremental concept formation.

1. INTRODUCTION

Software reuse is the process of creating software systems from existing artifacts rather than building them from scratch in order to enhance the productivity and the quality of software. During the development of a new software system, the software reuse process consists mainly into (1) building reusable artifacts, and (2) reusing these artifacts, i.e., retrieving, evaluating and adapting existing ones to the new requirements. The artifacts to be reused can be requirement definitions, source code fragments, design structures, specifications, manual descriptions, graphical data models, test cases, process management data and so on (Freeman, 1987; Jones, 1984; Krueger, 1992).

The success of the software reuse process (Berzins & Luqi, 1991) depends on many criteria such as (1) the existence of a software base or repository containing many instances of each type of software artifacts, and the availability of incremental procedures for updating the repository, (2) the availability of efficient algorithms for describing and indexing artifacts, and for finding appropriate artifacts from a possibly large software base, and (3) the reusability of existing artifacts (according to their original design, to the relevance of their specifications and possible reuse metrics, etc.).

In preparing artifacts for reuse, there are three main activities:

(1) choosing artifacts with high reuse potential,

(2) describing these artifacts,

(3) cataloging the artifacts in a repository.

These activities are globally characterized as *abstracting* in (Krueger, 1992), since they are closely connected and are not necessarily done separately. The authors in (Basili, Caldiera & Cantone, 1992) emphasize the importance of these reuse packaging activities and propose a new software engineering organization separating these activities from the project specific activities. Concept formation methods such as the ones proposed in this paper may be useful for discovering better abstractions or simply for organizing the artifacts into a repository for retrieval purposes.

Concept formation methods are distinguished from conceptual clustering methods by their incremental nature (Gennari, Langley & Fisher, 1990). This is an important characteristic for practical use in operational environments where new artifacts are continually added to the repository. Conceptual clustering can be defined by the following problem:

- Input: given a set of instances with their associated descriptions;
- Output: find a concept hierarchy (classification structure) made of,
    . a set of clusters (classes) of instances;
    . an intensional description for each cluster;
    . a hierarchical organization of the clusters.

Here we consider building the concept hierarchy from a set of artifacts based on a concept lattice structure (also called Galois lattice) and variants thereof.

In (Wille, 1982), Wille introduced formal concept analysis which is based on generating the concept (or Galois) lattice of a binary relation R between a set of instances and a set of features. Initially, it was used in the fields of symbolic data analysis and artificial intelligence for knowledge representation and acquisition (Wille, 1992). The concept lattice can be considered as a concept hierarchy where each node of the lattice corresponds to a concept. As opposed to other concept formation methods, the generated concept hierarchy has well-defined semantics and does not depend on some parameters, input ordering or algorithm specifics. This is particularly important for the users who feel more comfortable with an automatically generated hierarchy if they can understand exactly why a given concept has been generated. Many algorithms have been proposed for generating the lattice and incremental algorithms have been proposed for generating the concept lattice and its corresponding Hasse diagram (Carpineto & Romano, 1993; Godin, Missaoui & Alaoui, 1991). In the same vein as other concept formation methods such as UNIMEM (Lebowitz, 1987) and COBWEB (Fisher, 1987), the new instances are integrated into the existing concept hierarchy using a top-down search of the lattice, and generating new classes (nodes) whenever necessary. Under the assumption of an upper bound on the number of features that describe an instance, the number of nodes (classes) that the algorithm produces has a worst-case linear growth with respect to the number of instances (Godin et al., 1991). Even though the algorithm is incremental, the resulting hierarchy does not depend on the order of introduction of new instances, as opposed to UNIMEM and

COBWEB. Another important distinguishing feature is that the hierarchy is not constrained to be a tree as in the preceding methods.

Another application area is using the concept lattice as a browsing space for information retrieval (Godin, Missaoui & April, 1993). In this context, the concept lattice is generated from the usual binary relationship between documents and indexing terms. The Hasse diagram is used as the basic structure supporting a browsing interface that permits gradual enlargement or refinement of the user's query expressed in terms of document and term subsets present in the lattice. In this paper, we consider applying this idea to retrieval of artifacts for reuse purposes. Browsing retrieval mechanisms are important for reuse because there is not always an exact match between the query and artifact's description of the repository either because the user does not use the same terms or because there is no artifact exactly matching the need. In this case the reuser wants to look around for an artifact that is close enough to be adapted to the reusers need. The hierarchy generated from the concept lattice can be used for browsing and understanding the relationships between the artifacts.

The LaSSIE system (Devandu, Brachman, Selfridge & Ballard, 1991) also advocates the use of a classification and browsing for reuse. In their system however, the classification is knowledge-based and has to be constructed manually. In (Maarek, Berry & Kaiser, 1991) traditional clustering methods are used on term phrases automatically extracted from text descriptions of software components to produce a classification for browsing purposes. The advantage of using a concept hierarchy such as we propose is that the intensional descriptions of the clusters can be presented to the user and this greatly enhances the usefulness of the hierarchy for browsing purposes. In (Gibbs, Tsichritzis, Casais, Nierstrasz & Pintado, 1990), the authors advocate the use of browsing tools for class exploration. They present the concept of affinity browsing where the similarity between classes may be based on the common interface. The *affinity browser* implements this concept by showing the classes within a user defined similarity threshold with respect to the current selected class. Wilde and Huitt (1992) suggest that the maintenance environment of an object-oriented system could provide clustering on several object features possibly user-defined for system exploration. Automatic generation of the interface hierarchy of a set of classes based on the theory of concept lattices is proposed in (Godin & Mili, 1993) for design purposes when building and maintaining class hierarchies. The hierarchy may also be used for browsing

from a perspective that is different from the inheritance hierarchy and is closer to the client's view of the library. This is similar in spirit to the work of (Oosthuizen, Bekker & Avenant, 1992). In their work they consider generating a structure very similar to a Galois lattice using several class features.

In (Mineau & Godin, 1994), an extension and a refinement of the concept lattice is proposed to deal with a richer representation language. The instances are represented by conceptual graphs, which is a well-defined and established knowledge representation formalism. This method has been proposed as a tool for induction of generic data models represented by graphs (Mineau, Godin & Missaoui, 1993). The hierarchy generated, called a knowledge space, is not necessarily a tree as in UNIMEM and COBWEB, or a lattice as in concept analysis. This difference is due to the fact that only a subset of the concept lattice is generated to cope with the additional complexity of dealing with a richer description language. This subset is characterized by the fact that the non-inherited part of any description can not be empty. The simplifying heuristic used for the KS is also considered in this paper for the context of simpler descriptions. In similar work, Castano and De Antonellis (1993) consider generating reusable Generic Conceptual Units from conceptual schemas. They use a two step process where the schemas are first clustered using traditional statistical methods and the Generic Units are abstracted from the clusters. Other related work based on traditional clustering methods include analyzing system modularization where the clustering of the system parts is produced using data bindings based on the actual code (Hutchens & Basili, 1993) or on module specifications (Carrington, Duke, Hayes & Welsh, 1993). The advantage of concept formation methods is that the clustering and generation of abstractions are tightly coupled within the same process. The clusters are therefore chosen based on their potential for producing good abstractions.

In this article, we describe several applications of concept formation methods based on concept lattices which are part of a major software engineering research project called the IT Macroscope Project, managed by the DMR Group Inc., involving the "Centre de Recherche en Informatique de Montréal" and many industrial and academic partners. Section 2 gives the basic definitions related to the concept formation methods. Section 3 illustrates how the methods are exploited for software reuse purposes, using real life applications supplied by the industrial partners involved in the IT Macroscope project.

2. CONCEPT FORMATION USING CONCEPT LATTICES AND VARIANT STRUCTURES

This section presents the basic concepts of concept lattice and pruned concept structures used for concept formation.

2.1 CONCEPT LATTICE

This subsection recalls basic definitions related to the concept lattice for a binary relation. More details are found in (Barbut & Monjardet, 1970; Wille, 1982). Given two finite sets E and E', and a binary relation R (Figure 1) between these two sets, there is a unique ordered set which describes the inherent lattice structure defining natural groupings and relationships among the elements of E and E' (Figure 2). This structure is known as a *concept lattice* (Wille, 1982) or *Galois lattice* (Barbut & Monjardet, 1970). Each element of the lattice is a couple (called concept by Wille (1992)), noted (X,X'), composed of a set $X \in P(E)$ and a set $X' \in P(E')$, where P(S) is the powerset of S. Each couple must be a *complete couple* as defined in the following.

A couple (X,X') from P(E) x P(E') is *complete* with respect to R if the two following properties are satisfied:

1) $X' = f(X)$ where $f(X) = \{x' \in E' \mid \forall x \in X, xRx'\}$.
2) $X = f'(X')$ where $f'(X') = \{x \in E \mid \forall x' \in X', xRx'\}$.

The couple of functions (f,f') is a *Galois connection* between P(E) and P(E') and *the concept (or Galois) lattice* G for the binary relation is the set of all complete couples (Barbut & Monjardet, 1970) with the following partial order: given $C_1=(X_1,X'_1)$ and $C_2=(X_2,X'_2)$,

$$C1 < C2 \iff X'_1 \subset X'_2 .$$

There is a dual relationship between the X and X' sets in the lattice, i.e.,

$$X'_1 \subset X'_2 \iff X_2 \subset X_1$$

and therefore,

$$C1 < C2 \iff X_2 \subset X_1.$$

The partial order is used to generate the graph in the following way: there is an edge (C_1,C_2) if $C_1 < C_2$ and there is no other element C_3 in the lattice such that $C_1 < C_3 < C_2$. C_1 is called *parent* of C_2 and C_2 *child* of C_1. The graph is usually called a *Hasse diagram* . By convention, when drawing a Hasse diagram, the edge direction is always upwards or downwards. The Hasse diagram represents the generalization/specialization relationship between the couples where $C_1 < C_2$ means that C_1 is more general than C_2. Given, C, a set of elements from G, inf(C) and sup(C) will denote respectively the greatest lower bound and lowest upper bound of the elements in C.

Complexity of the concept lattice

It is important to note that taking all possible object subsets produces an exponential number of nodes. However, when there is a fixed upper bound, K, on $\|f(\{x\})\|$, which is usually the case in practical applications, the worst case complexity of the structure is linearly bounded with respect to $\|E\| = n$ (Godin, Saunders & Gecsei, 1986):

$$\|G\| \leq 2^K n.$$

The upper bound on $\|G\|$ is exponential in K; however, experience with real applications and theoretical results with randomly assigned elements show that in practice, $\|G\|/n$ is fairly stable and much smaller than this upper bound (Godin et al., 1993).

Another important point for practical use of the lattice is that algorithms for incrementally updating the structure by adding new documents one at a time have been devised. Empirical data from several applications show that adding a new document is made in **O**(n) time and disk I/Os (Godin et al., 1991). With the hypothesis of a fixed upper bound for the number of index terms per document, this is also confirmed by a complexity analysis of the algorithm.

2.2 INHERITANCE CONCEPT LATTICE

There is much redundant information in a concept lattice. For a couple C= (X, X'), X will be present in every ancestor of C and symmetrically, X' will appear in every descendant. The inherited (redundant) elements may therefore be eliminated without losing any information. For a couple C = (X, X'), let X'' be the non-redundant elements in X, and X''' the non redundant elements in X'. Formally, given a complete couple C = (X, X'):

$X'' = \{x \in E \mid x \in f'(X')$ and there is no other couple $C' = (Y,Y') > C$ such that $x \in Y\}$

$X''' = \{x' \in E' \mid x' \in f(X)$ and there is no other couple $C' = (Y,Y') < C$ such that $x' \in Y'\}$

The following are equivalent definitions making explicit the fact that an element appears in X'' if it is in the most general couple containing it in G, and an element appears in X''' if it is in the most specific couple containing it in G:

$X'' = \{x \in E \mid x \in f'(X')$ and $C = \sup \{C' = (Y,Y') \in G \mid x \in Y\}\}$

$X''' = \{x' \in E' \mid x' \in f(X)$ and $C = \inf \{C' = (Y,Y') \in G \mid x' \in Y'\}\}$.

It should be noted that an element is in X'' if it is related to the elements of X' and no other element. In other words X'' is related exactly to the elements of X' and symmetrically, X''' is related exactly to the elements of X. Therefore:

$X'' = \{x \in E \mid f(\{x\}) = X'\}$

$X''' = \{x' \in E' \mid f'(\{x'\}) = X\}$.

An *X-inheritance concept lattice* is defined as the set of couples (X'', X') (Figure 3), an *X'-inheritance concept lattice* is defined as the set of couples (X, X''') (Figure 4) and an *inheritance concept lattice* is defined as the set of couples (X'', X''') (Figure 5). Given a couple C, the corresponding values of X (respectively X') can be

computed by taking the union of the X'' (respectively X''') sets for the descendants (respectively ancestors) of C, including C itself.

Using the previous definitions, an X-inheritance concept lattice is expressed as the set of couples (X'', X') such that there is a complete couple (X,X') and:

$$X'' = \{x \in E \mid f(\{x\}) = X'\}$$

An X'-inheritance concept lattice is defined as the set of couples (X,X''') such that there is a complete couple (X,X') and:

$$X''' = \{x' \in E' \mid f'(\{x'\}) = X\}.$$

An inheritance concept lattice is defined as the set of couples (X'',X''') such that there is a complete couple (X,X') and:

1) $X'' = \{x \in E \mid f(\{x\}) = X'\}$

2) $X''' = \{x' \in E' \mid f'(\{x'\}) = X\}$.

Depending on one hand on the application and, on the other hand on time versus space performance constraints, these structures may be considered as an alternative to the full concept lattice representation. These definitions will also be used as a basis for the definition of the pruned concept lattice introduced in the next section.

2.3 PRUNED CONCEPT LATTICE

For some applications, the concept lattice contains too many nodes and a well chosen subset might be more useful. One such subset is called an X'-pruned concept lattice and has been initially used in a more general context for conceptual clustering of conceptual graphs (Mineau & Godin, 1994). The following presents the concept in the simpler context of a binary relation.

The X'-pruned inheritance concept lattice (PICL) for the binary relation of Figure 1 may be generated from the X'-inheritance lattice of Figure 4 by eliminating the couples other than the infimum which have *empty X''' sets*. For example the couple, ({1,2},Ø), would be eliminated. The resulting PICL is shown in Figure 6. Formally, the PICL is the set of couples (X, X''') that satisfy:

1) $X = f'(X''')$

2) $X''' = \{x' \in E' \mid f'(\{x'\}) = X\}$.

It is easy to show that the number of couples in the PICL is bounded by ||E'||. Given the upper bound K on ||f({x})||, ||E'|| is bounded by K ||E|| and therefore:

$$||PICL|| \leq K\ ||E||.$$

In most applications the actual growth would tend to be asymptotically logarithmic in ||E|| as suggested by the Bradford-Zipf law (Brookes, 1977). This law formalizes a well-known phenomenon applicable to the growth of a vocabulary with respect to the length of a text, or the growth of an indexing vocabulary with respect to the number of documents.

The number of ancestors for a node is bounded by K, because each element of f({x}) appears in one node and every ancestor of a node containing x contains at least one element of f({x}). This is therefore true for the number of parents:

$$||parents|| \leq ||ancestors|| \leq K.$$

3. APPLICATIONS

To test the concept formation methods based on the structures introduced in the previous section, incremental algorithms have been devised and implemented in C on a SUN platform. A user interface was developped in Smalltalk within the environment of ObjectWorks® (release 4.0). In the following, we show how our concept formation schemes have been exploited for software reuse in two ways: as a navigational space for *retrieval*, and for *reuse packaging* activities by suggesting better abstractions. We will illustrate these ideas using data from the IT Macroscope project, which was provided by the National Bank of Canada and the DMR Group.

3.1 RETRIEVAL

Traditional clustering techniques have been used for building and browsing a hierarchy of software components (Maarek et al., 1991). The advantage of using the concept formation methods previously described is that direct querying using the index terms and browsing are easily combined in a natural manner within the same structure (Godin et al., 1993).

To illustrate the retrieval process, an application taken from the data dictionary of the National Bank of Canada is used. This dictionary is considered as a standard reference for building database schemas and ensuring uniformity and standardization within the companies applications. When a new application is defined, the analyst starts by determining if existing data elements satisfy the needs of the application. This is a simple form of reuse but the same retrieval problems and strategies are applicable when reusing other types of artifacts. Given the large volume of data element definitions, finding a data element is a difficult process. To assist this task, controlled index terms are assigned to the data elements by the data administration personnel. This indexing relation for a subset of 796 data elements was used to generate the concept lattice and the X'-PICL, and is used for illustating the retrieval process. Figures 7-8-9 and 7-10-11 respectively show screen dumps from two sequences of interactions which might have been produced by a user looking for a data element giving "the date when the address change of a client becomes effective". Figures 7-8-9 would be produced using the full

concept lattice while Figures 7-10-11 would be produced using the PICL[1]. Since the data is in French, translations into English are provided to explain the examples.

Browsing is based on jumping from one couple (node) to another in the structure. The screen shows the *current couple* in a large box in the middle of the window. The parents are displayed above and the children below, in smaller boxes. To avoid redundancy, the terms of the current node are not repeated in the children nodes for the concept lattice. To go from one couple to another, the user may simply select a parent or child by clicking with the mouse in the corresponding box. For example in the sequence of Figures 7 to 9, the user has successively selected the boxes corresponding to the terms: "DATE" in Figure 7, "AAAAMMJJ" in Figure 8, then, "EFFECTIVE" in the following screen that is not reproduced here which results in Figure 9. To find the right element the user could continue selecting the child "ADRESSE" (French for *address*) in Figure 9, then, "CLIENT" and finally "CHANGEMENT"[2] (French for *change*). One important aspect of this process is that the user does not have to know what are the "right" query terms. He only has to select among the candidate terms those which seem the most appropriate. In the current system, the users do retrieval by Boolean querying. If the user does not use the right term, the system returns nothing in response. In this case the user might think, there is no related data element and decide to create a new redundant element. Our analysis of the current system shows that this has been the case quite often in the past. Although this is not the primary purpose of the tool, the users have noticed several such occurrences simply by browsing through the hierarchies which show related elements close to each other and closer examination of these revealed that they represented the same thing.

Direct querying is also supported by the tool. By clicking on the appropriate icon, the user then supplies the terms and the system brings the focus on the most general couple containing these terms. For example, to arrive at the state in Figure 9, the user could have specified directly the terms "EFFECTIVE", "DATE" and "AAAAMMJJ" without following the path in the lattice. In fact if he had only specified the term "EFFECTIVE", the result would have been the same because the most general node containing "EFFECTIVE" also contains the two other keywords "DATE" and "AAAAMMJJ". This is apparent in Figure 9 because there is no parent of the

[1] Figure 7 shows the starting screen which is the same for both the concept lattice and PICL.
[2] The corresponding English terms for "addresse" and "changements" are address and change respectively. "AAAMMJJ" represents the year-month-day format.

current node that contains "EFFECTIVE". The interface supports many additional functions but we limit our presentation to the most essential ones.

The differences between the two alternative structures is apparent when examining the two interactions in parallel. The top part of the structures are quite similar and the beginning three steps of the interaction is therefore similar: slection of the children containing "DATE", then "AAAAMMJJ", then "EFFECTIVE". The first screen (Figure 7) is identical for the two structures. After selecting "DATE", the resulting screens (Figure s 8 for concept lattice and 10 for PICL) start to be different but the selection "AAAAMMJJ" which is the date format can be done in both cases. However in the forth step (Figures 9 and 11), the structure becomes more abrupt for the PICL since many intermediate nodes having null non-inherited parts have disappeared and no further refinement for the example query is appropriate. The user then has to exhaustively examine the list of data elements to find the correct one. This is done by clicking on the current node or by selecting the right icon. A new window appears, entitled "Liste des objets" in Figure 11, showing a scrollable list of data element identifiers for the current couple in the left pane. When clicking on an element, the right pane displays the description of the element. In this case this description is limited to the set of related index terms. In contrast, the concept lattice structure allows the user to further refine his query step by step by making three additionnal selections: "ADRESSE", then "CLIENT" and finally "CHANGEMENT". The additional richness of the concept lattice supports smoother browsing at the cost of additional computing resources. To better unserstand the tradeoffs the relative computational cost of the two approaches is examined in section 4.

Another experiment was conducted on the text of DMR Productivity Plus™ system development methodology. The goal is to help locating parts of the methodology for specific tasks such as exploration, update and learning. The text was therefore divided into sections and automatic indexing techniques were applied to generate index terms for each section. The automatic indexing process is a combination of sophisticated natural language analysis and statistical techniques.

The natural language analysis was performed by the Termino system developed at the Centre ATO-CI (Faraj, Godin, Missaoui, David & Plante, 1994). Termino does a morpho-syntactic analysis of french texts and extracts a set of lemmatized linguistic units categorized into several classes such as nouns, verbs, adjectives and phrases. As opposed to approaches based on cooccurrences, the phrases are based on the recognition of the syntactic relationships between the words in the text. Although syntactic phrase indexing has not given significant improvements over simpler statistical cooccurrence in previous work on english texts (Fagan, 1989), the richer syntactic structures of french texts provide more clues for correct parsing of texts in the absence of semantic analysis. From this set of units, traditional statistical methods (Salton & Buckley, 1988) were applied to derive significant index terms. There are many possible combinations for the choice of unit categories and weighting methods and there is no clear answer to the best strategy to use. One combination which has shown interesting results is to combine nouns and syntactic noun phrases together and using the well-known *tf.idf* weighting method on these units and keeping the most significant units using a threshold over the computed weight. Although no formal evaluation has been performed, the following example shows the potential of the method. Figure 12 is a sample screen produced by generating the PICL from this indexing method. The current couple corresponds to the term model ("modèle" in French). The screen shows sub-concepts corresponding to conceptual model ("modèle conceptuel"), definition of a model ("définition du modèle"), level ("niveau"), physical model ("modèle physique") and functional model ("modèle fonctionnel"). Only the *level* concept might seem unfamiliar, however it is very significant for the P^+ methodology which advocates the use of multi-level models going from general to specific modelling tasks. This tool can, not only, be used for retrieval purposes but is also used as a tool for building a knowledge representation basis of the methodology. More research in this direction is currently being pursued.

3.2 REUSE PACKAGING

Concept formation methods may also help in the reuse packaging activities by suggesting better abstractions for reuse purposes. In one experiment in the context of the IT Macroscope project, this was applied to the repository of the National Bank of Canada. The experiment we present here concerns a repository of entity

definitions in terms of their attributes. The attributes are part of the data dictionary described in the previous section. The entities are listed in a flat manner without representing common parts explicitly as is done with semantic data models. The resulting concept hierarchy generated from this kind of input help identify more generic entities from existing ones. This is a key principle underlying reuse where more generic artifacts have better reuse potential.

As an illustration of this idea, Figure 13 shows part of the concept lattice, and Figure 14, the corresponding part of the PICL. In both cases, the *Deposit products* concept represents a set of 13 entities which share the four attributes "PDSEQP", "PDCLID", "PDSCID" and "PDDEVP". These attributes are always necessary for financial products where money can be deposited, such as normal checking accounts and long term deposit accounts. This commonality should therefore be represented explicitly in the repository as a more general entity with higher reuse potential. Providing this new representation helps the analyst in understanding the relationships between the entities and provides him with more abstract entities with better reuse potential. A sub-concept of this concept is the deposit products with varying interest rates which have the two additional attributes "TITEFF" and "TIDEFF". The specialization relationship is explicitly represented in both structures. The example also shows the difference between the concept lattice and PICL with respect to the *deposit product with client class* that appears in the concept lattice. This concept has been eliminated in the PICL because there is no non-inherited attributes. Therefore, the *deposit products with penalties* would inherit directly from *deposit products* in the PICL representation. The deposit products with penalties share additional attributes related to the penalty associated with terminating a long term investment. As for the retrieval application, the additional richness and refinement of the concept lattice is to consider in the light of the additional computational cost.

In this application, the abstracted artifacts are quite simple in nature. However, the same general idea could be applied to more complex objects with richer descriptions (Mineau et al., 1993).

4. COMPLEXITY OF THE METHODS

This section compares the computational cost of the concept lattice and PICL concept formation methods as implemented in C on a SUN SPARC 1^+ workstation. This analysis is useful for understanding the practical tradeoffs between both structures. From a practical point of view two important constraints must be addressed. First, in most applications such as the ones described in section 3, it is necessary to maintain not only the concepts (couples) of the structure but also the *generalization/specialization relationships* between the concepts. Secondly, the algorithms need to be *incremental*. It is important to have the possibility of dynamically adding a new artifact by modifying the existing structure without having to regenerate it from scratch. We have therefore implemented incremental algorithms for building the concept lattice and the PICL. Details of the concept lattice algorithm are found in (Godin et al., 1991).

Two types of comparison are performed between the PICL and concept lattice (G) incremental algorithms: simulation and empirical tests using the data elements application. Figures 15 and 16 give the CPU time and regression analysis for the simulation of a binary relation based on a uniform distribution hypothesis using $||E'|| = 500$ and $k=6$ where k is the average of $||f(\{x\})||$, the number of features per object. The Figures give the total time in seconds for incrementally building the hierarchies. As shown by these experiments, building the PICL is clearly more efficient. The same general behavior is observed for other values of $||E'||$ and k. Much of the difference between the two algorithms can be explained by the difference in the number of elements in both structures. This is the main factor determining the complexity of the algorithms. Figure 17 shows the difference between $||G||$ and $||PICL||$ for $||E'|| = 500$ and $k=6$. Remember that $||PICL||$ is bounded by $||E'||$. As $||E||$ grows, this difference gets larger and the difference in CPU time follows.

The $O(||E||)$ average time complexity for adding an element is apparent for both algorithms. Given the $O(||E||)$ complexity for adding one element, the total time for adding $||E||$ elements should be $O(||E||^2)$. Figure 15 shows the results of a polynomial regression of order 2 for the incremental PICL algorithm. The same results for the concept lattice incremental algorithm appears in Figure 16.

Analysis of the performances on the data elements application reported in section 3.1 reveals the same general behavior. The CPU time and regression analysis are given in Figures 18-19 for the PICL and concept lattice. The number of elements for both structures appears in Figure 20. However, there is one unanswered question. The theoretical analysis for the PICL (see section 2.3) suggests that the complexity is in the worst case bounded by the number of elements, and in effect, as expected, we observe a sub-linear growth for ||PICL||. In fact, the regression in Figure 21 confirms the Bradford-Zipf law of asymptotically logarithmic growth. Only the values for ||E|| greater or equal to 500 are taken into account. Therefore, we should observe a ||E|| log ||E|| CPU time after that point for constructing the PICL. However the observations show a quadratic growth which may be due to the implementation. There is therefore reason to believe that better performances could be attained. This is not the case for the concept lattice which has linear growth as depicted in Figure 22.

CONCLUSION

In this paper, concept formation methods are exploited for various software reuse purposes such as building a navigational space for retrieval, and reusing packaging activities. As an illustration of the first reuse process activity, we showed how the concept formation methods can assist a user in his task of retrieving data elements from a data dictionary by browsing through the generated concept hierarchy. In another application, the concept formation methods serves to build a navigational space to locate parts of a textual document, once an automatic indexing method is applied to the original text. As an illustration of reuse packaging activities, we showed how the concept hierarchy built from a set of entities defined in terms of their attributes is used to find more generic entities from existing ones.

The concept lattice structure and one of its variants, called the PICL, were both used in the applications and relative merits were highlighted. Although the concept lattice may be more expressive and complete, the generation is shown to be more costly than the PICL variant. More formal experiments are needed to assess the relative usefulness of each structure.

As an extention of this work, we are currently exploring refinements of the concept formation methods used here to more handle more complex artifacts involving richer representations such as graphs (Mineau et al., 1993).

FIGURES

E'

	R	a	b	c	d	e	f	g	h	i
	1	1	0	1	0	0	1	0	1	0
	2	1	0	1	0	0	0	1	0	1
E	3	1	0	0	1	0	0	1	0	1
	4	0	1	1	0	0	1	0	1	0
	5	0	1	0	0	1	0	1	0	0

Figure 1. Matrix representation of a binary relation R.

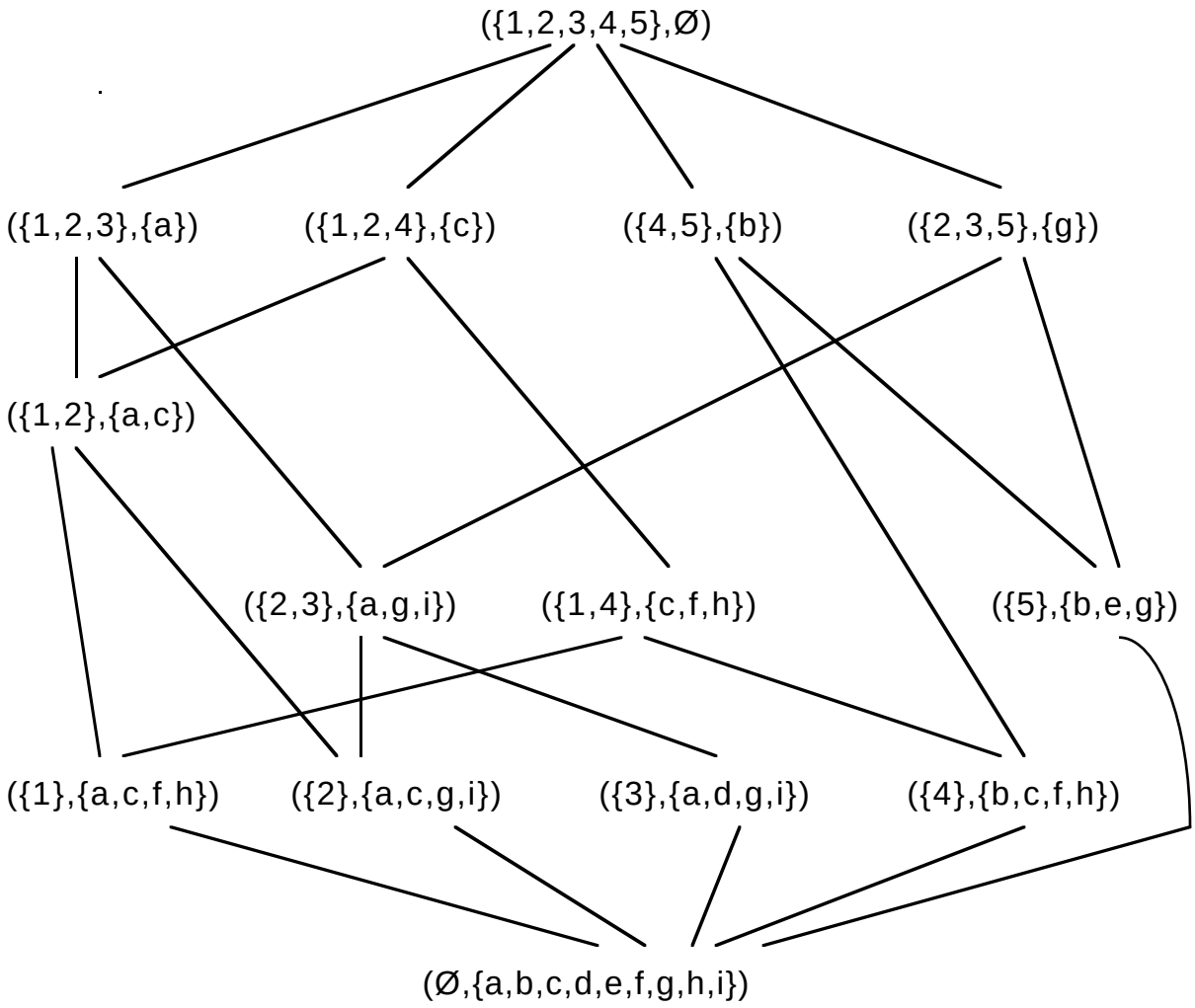

Figure 2. Concept lattice for the relation in Figure 1.

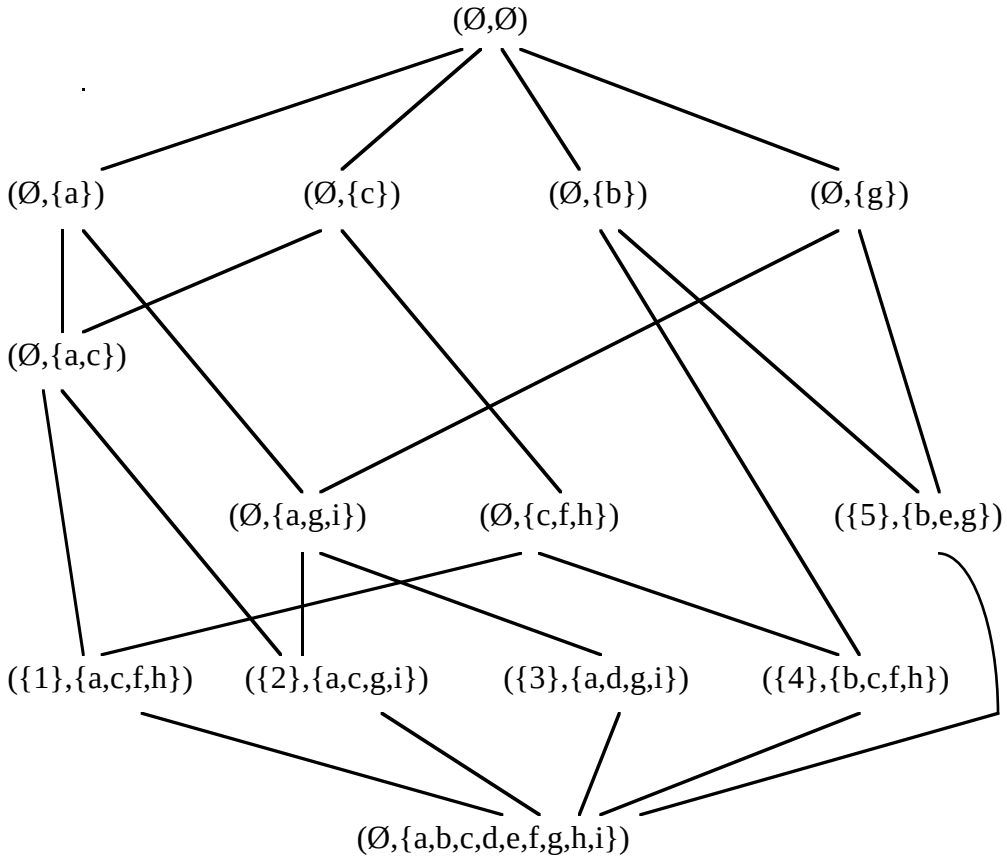

Figure 3. X-inheritance concept lattice.

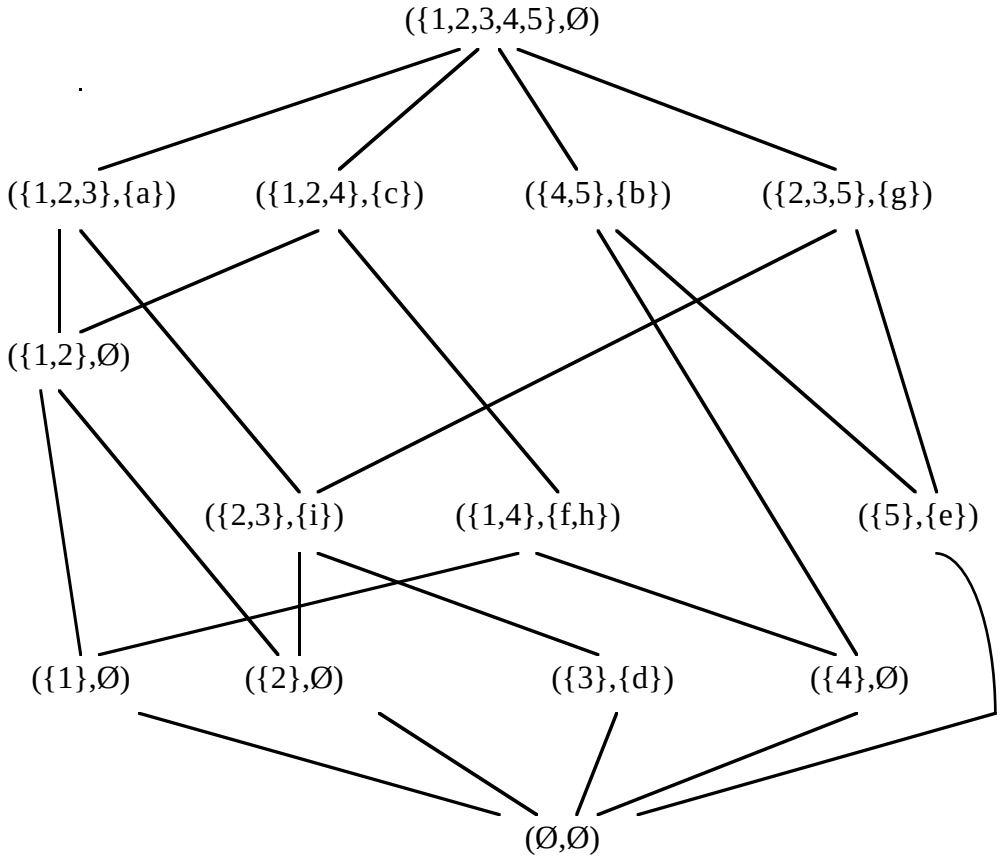

Figure 4. X'-inheritance concept lattice.

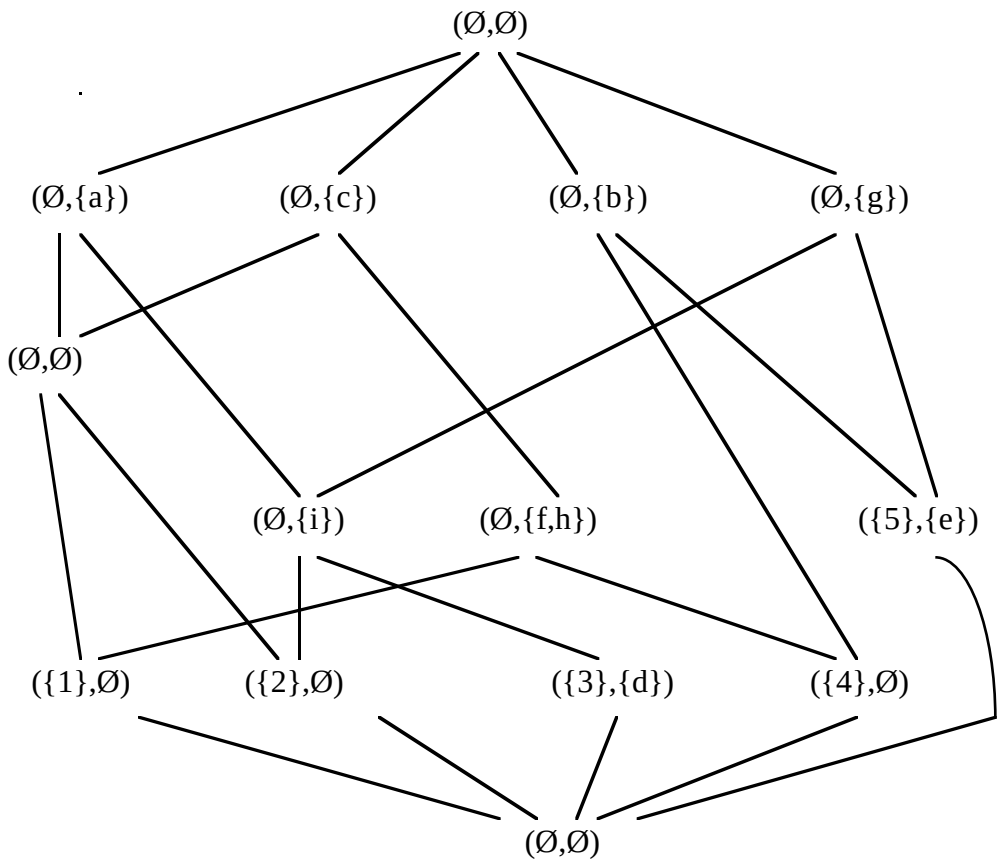

Figure 5. Inheritance concept lattice.

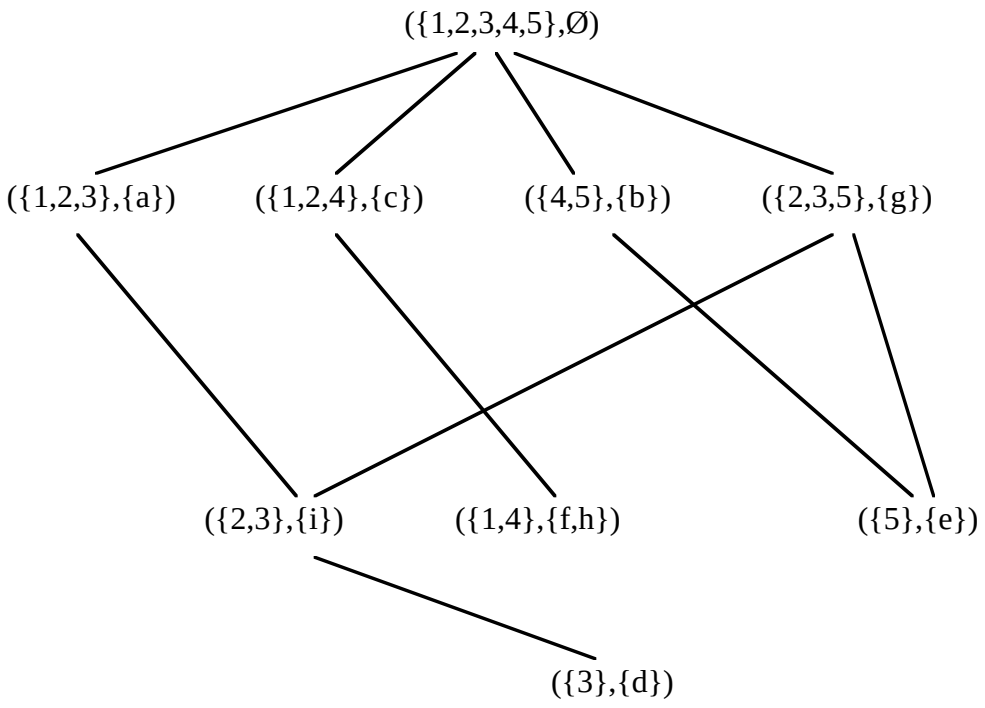

Figure 6. PICL for the relation in Figure 1.

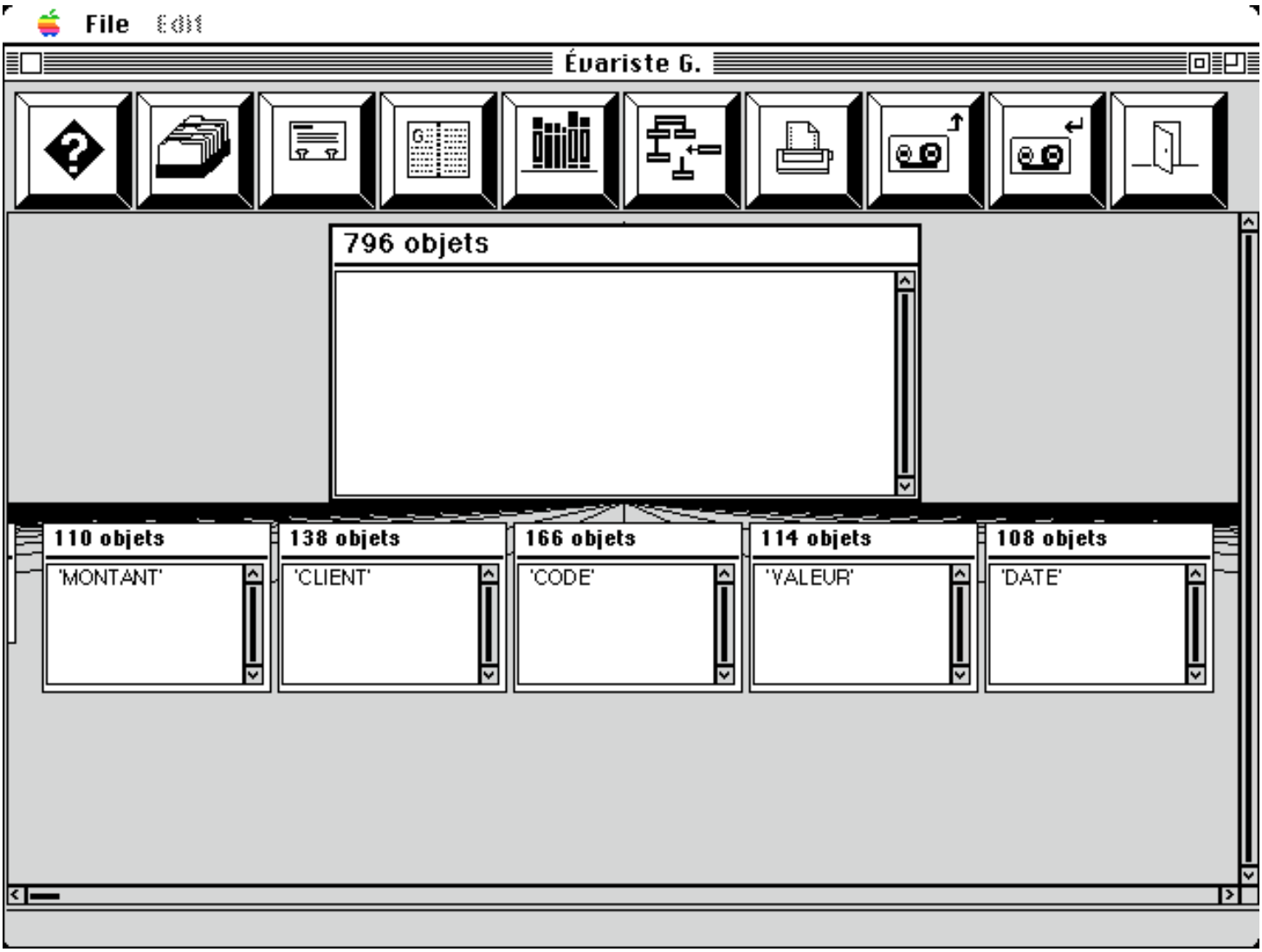

Figure 7. Initial screen for concept lattice or PICL.

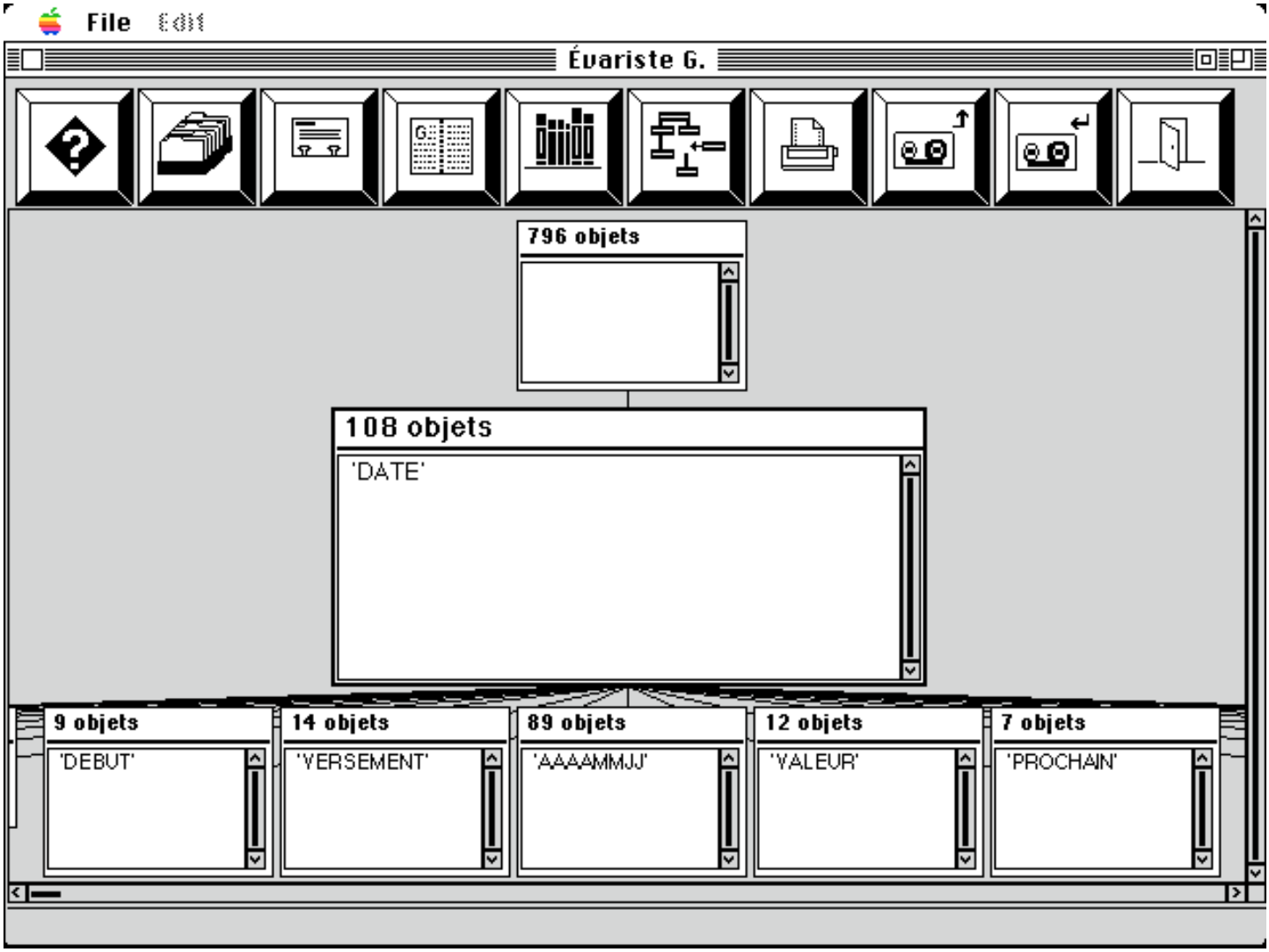

Figure 8. Result of selection of "DATE" from Figure 7 for concept lattice.

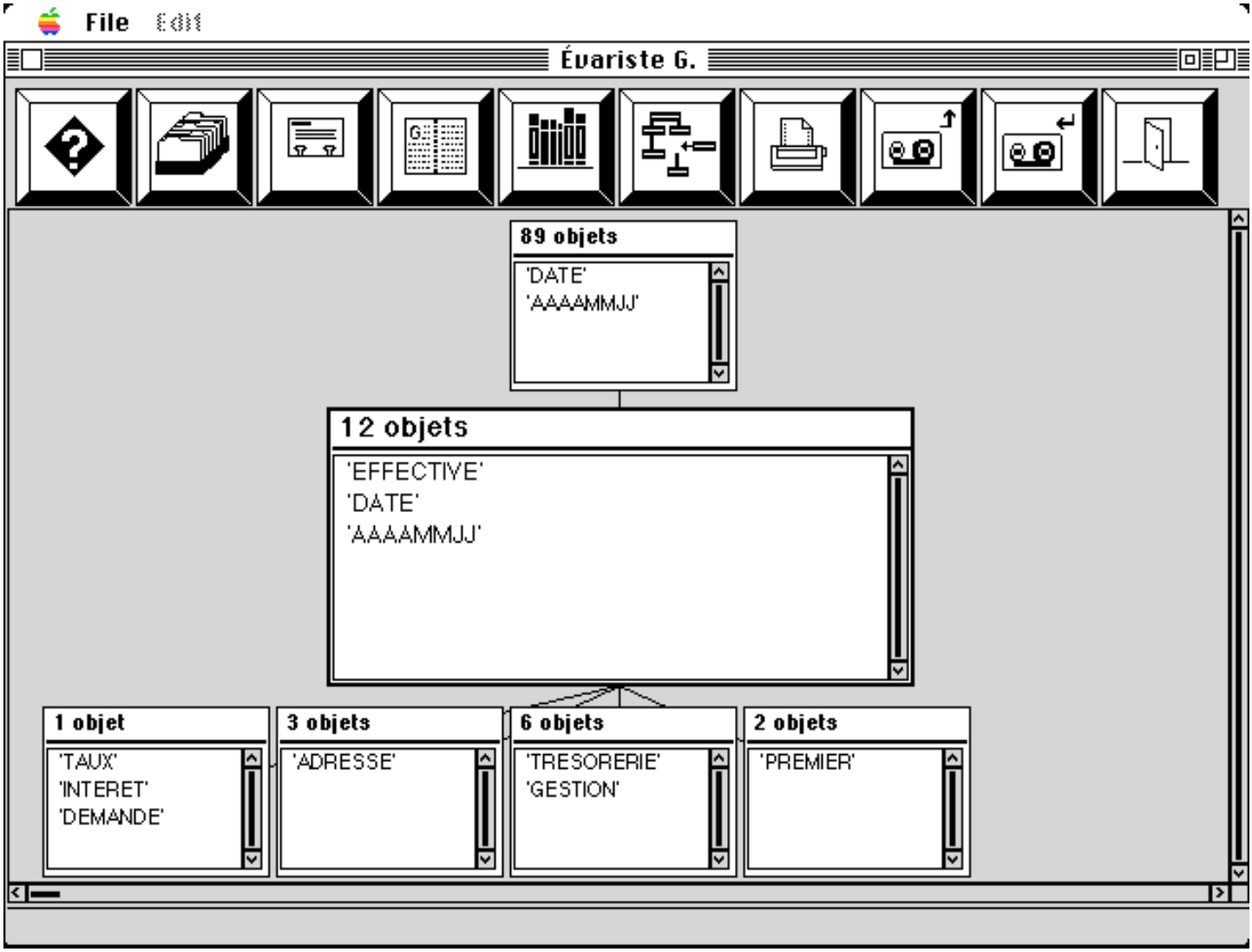

Figure 9. Result after selection of "AAAAMMJJ" then "EFFECTIVE" from Figure 8.

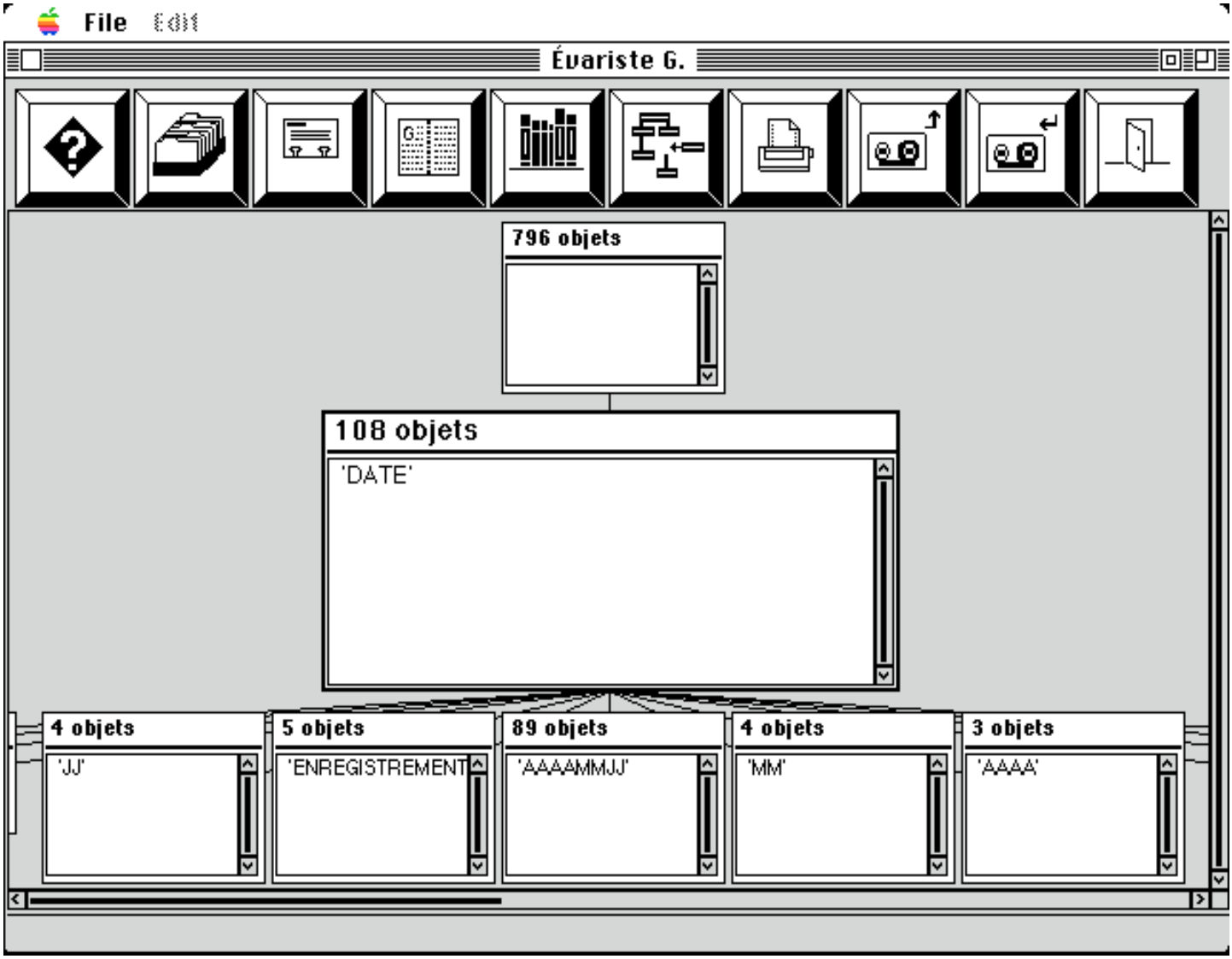

Figure 10. Result of selection of "DATE" from Figure 7 for PICL.

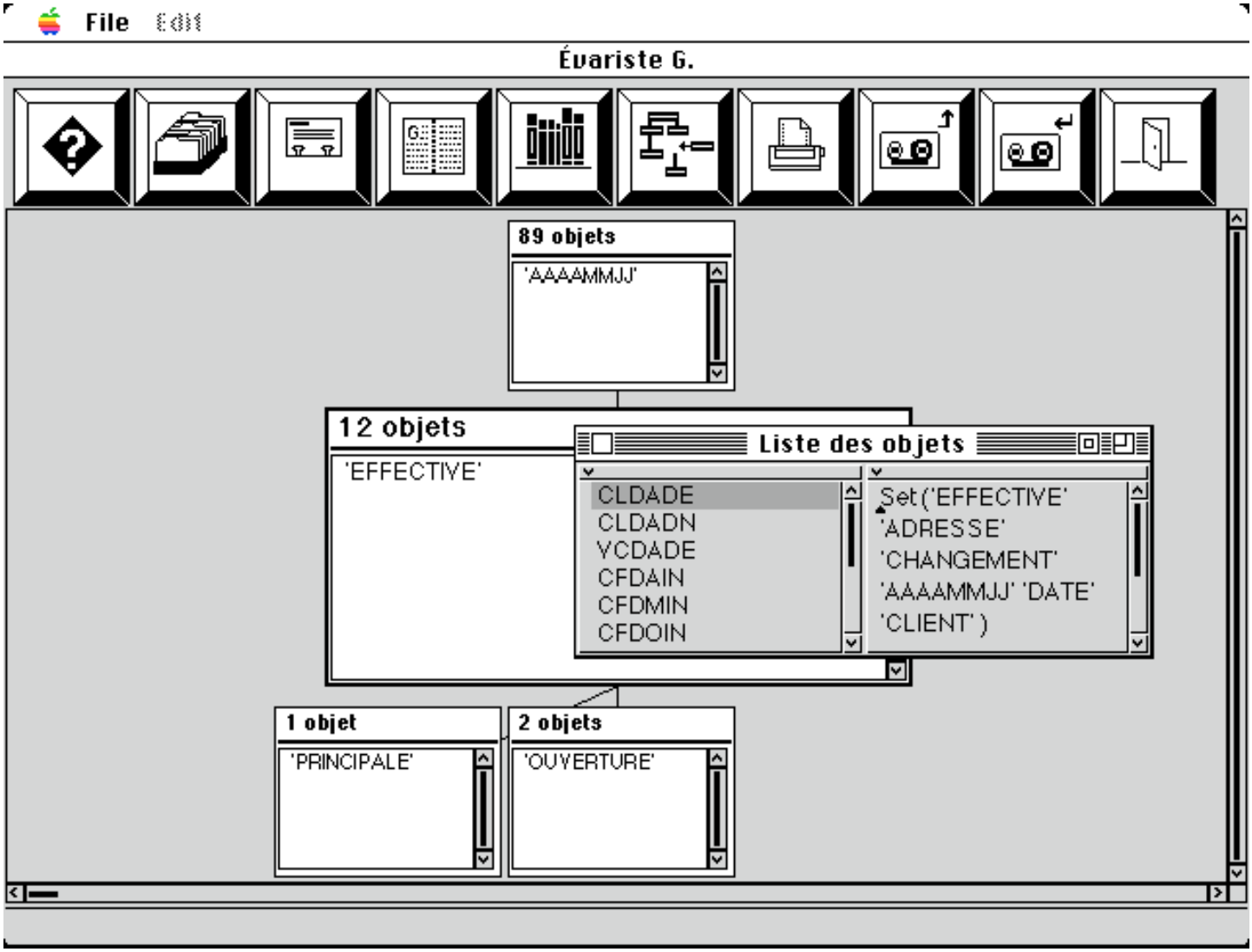

Figure 11. Result after selection of "AAAAMMJJ" then "EFFECTIVE" from Figure 10 and display of list of data elements for current couple.

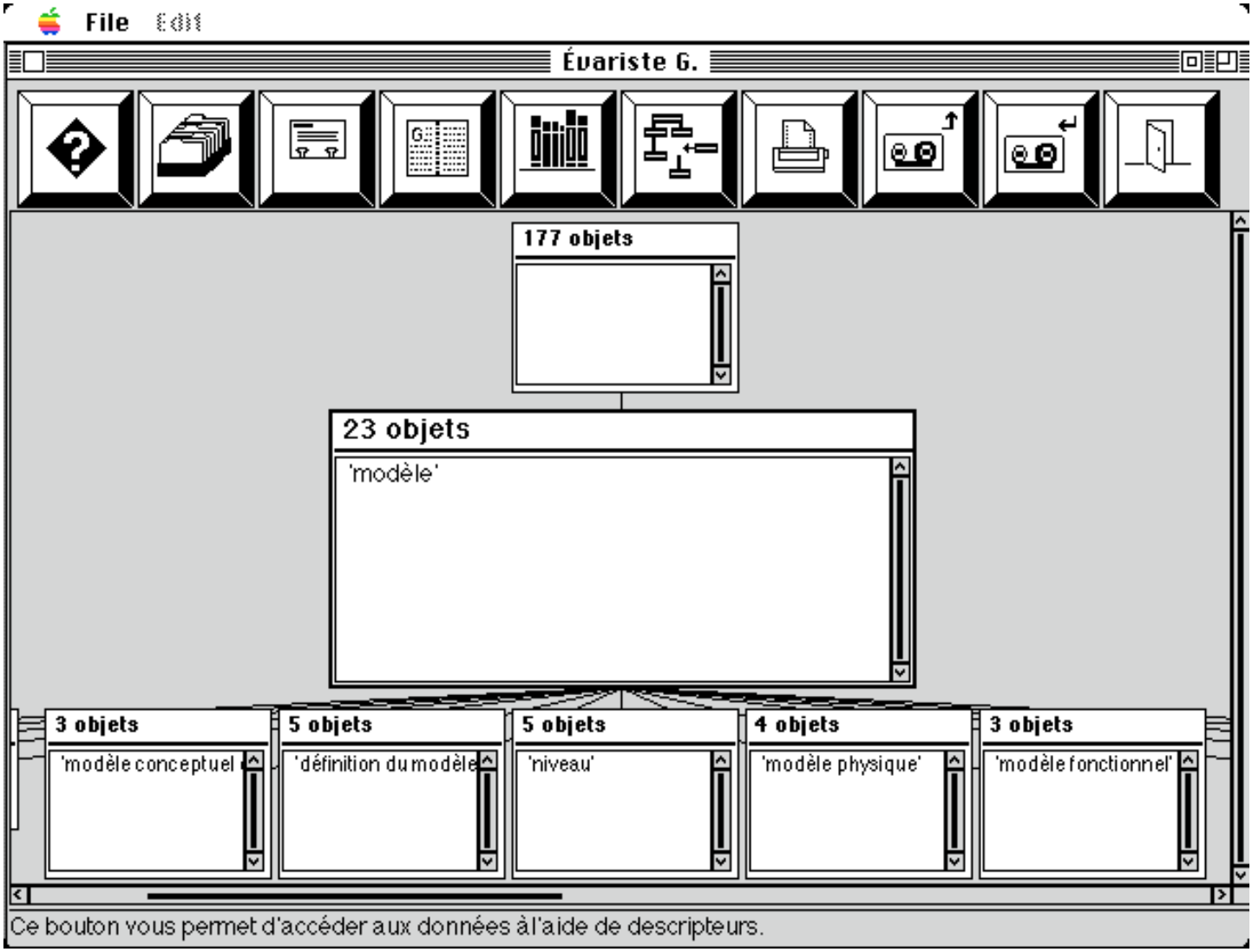

Figure 12. Sample screen from PICL generated using the methodology text.

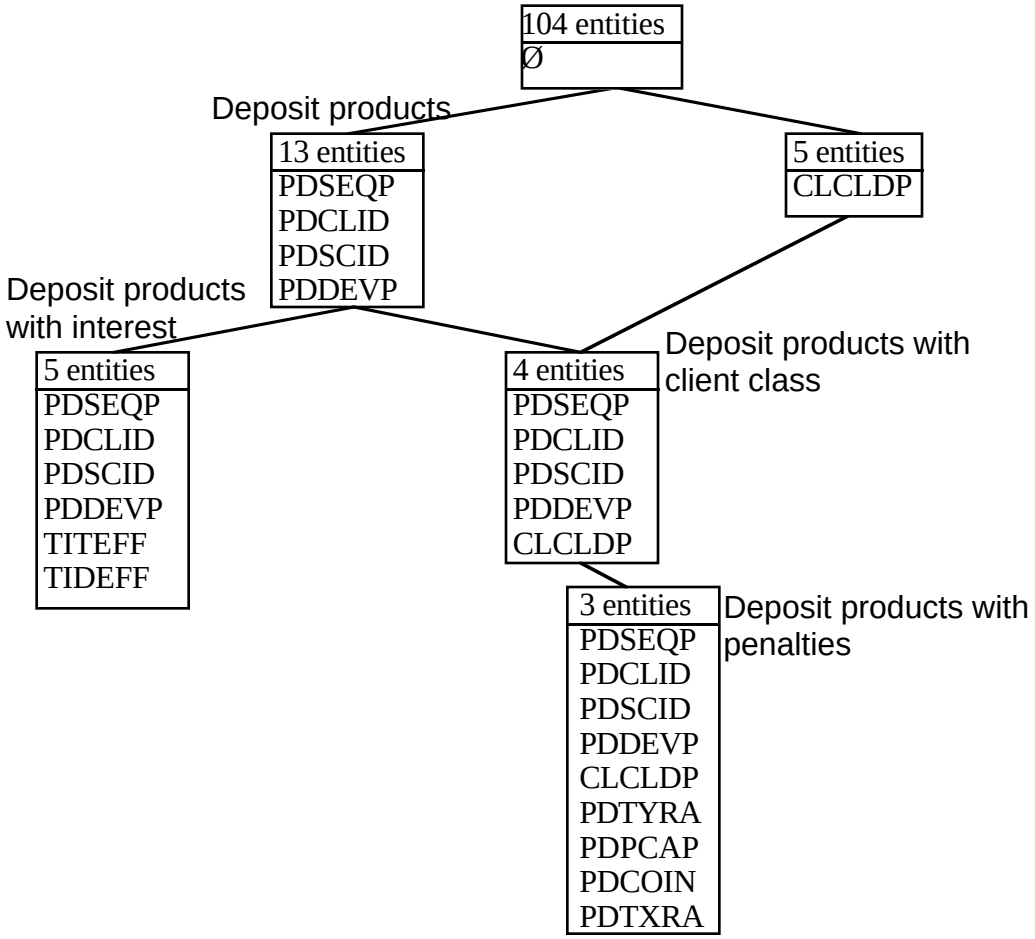

Figure 13. Part of concept lattice for entity-attribute relationship.

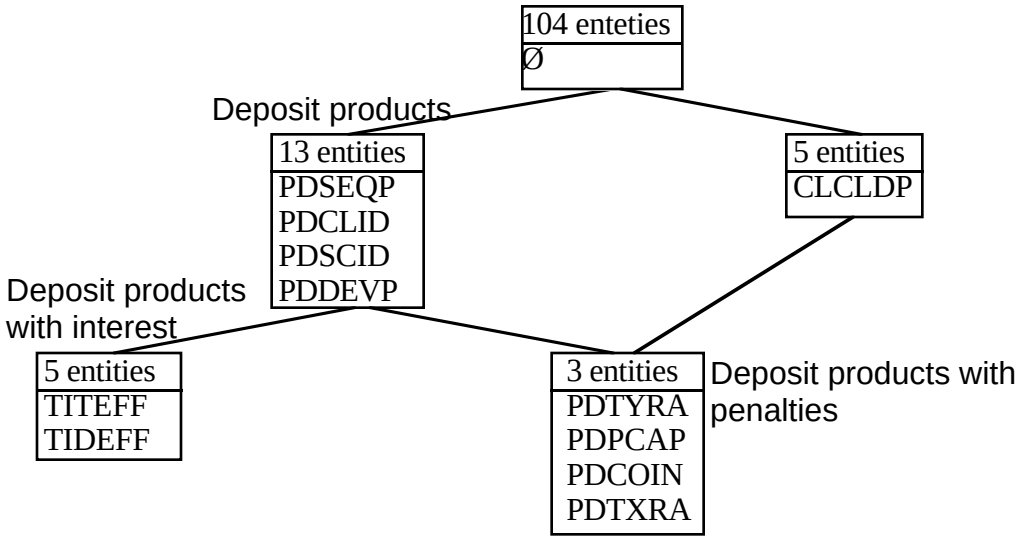

Figure 14. Part of PICL for entity-attribute relationship.

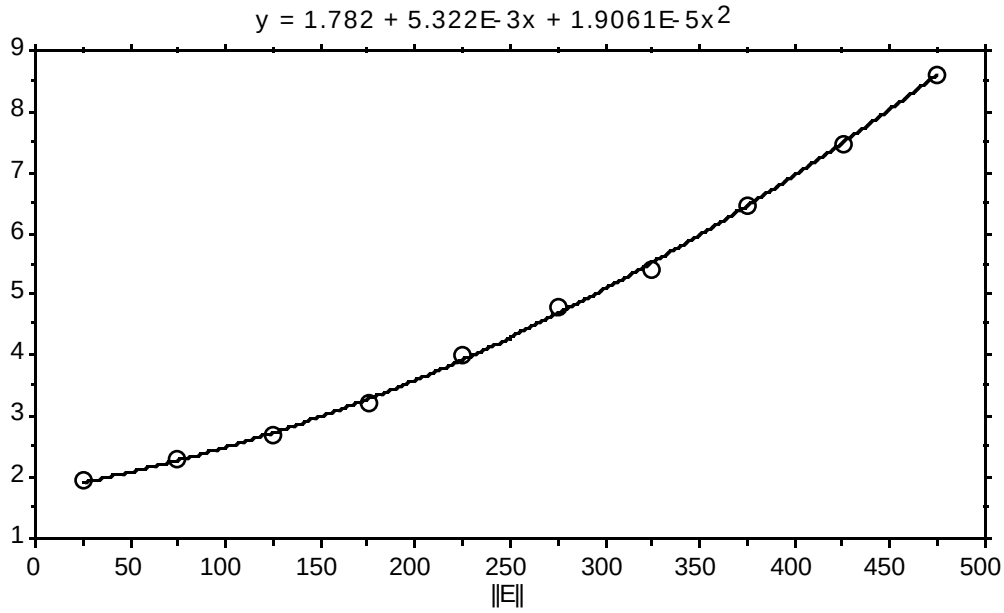

Figure 15. Regression for CPU time with ||E'|| =500 and k=6 using the incremental PICL algorithm.

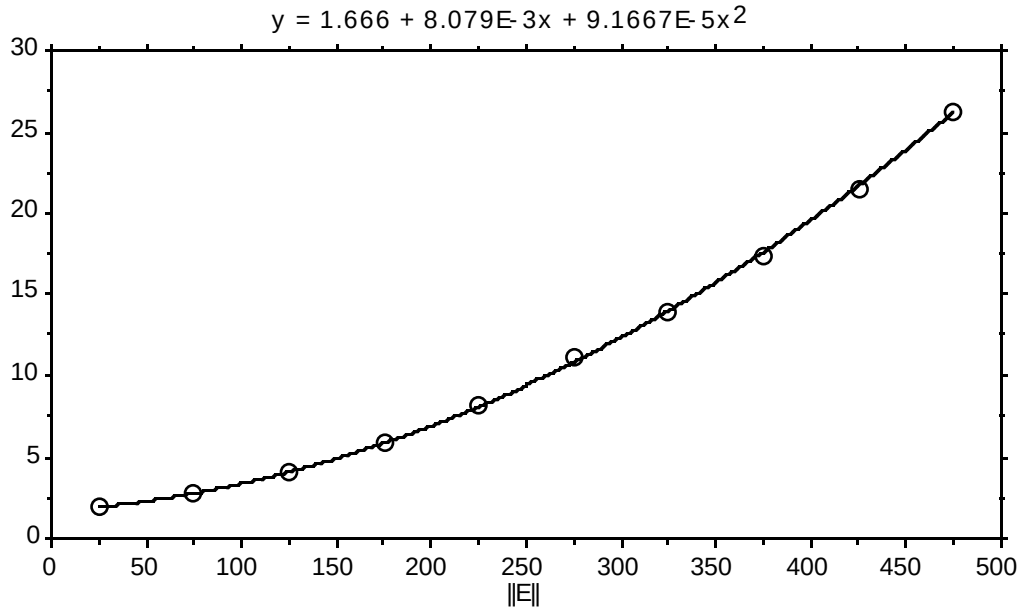

Figure 16. Regression for CPU time with ||E'|| =500 and k=6 using the incremental concept lattice algorithm.

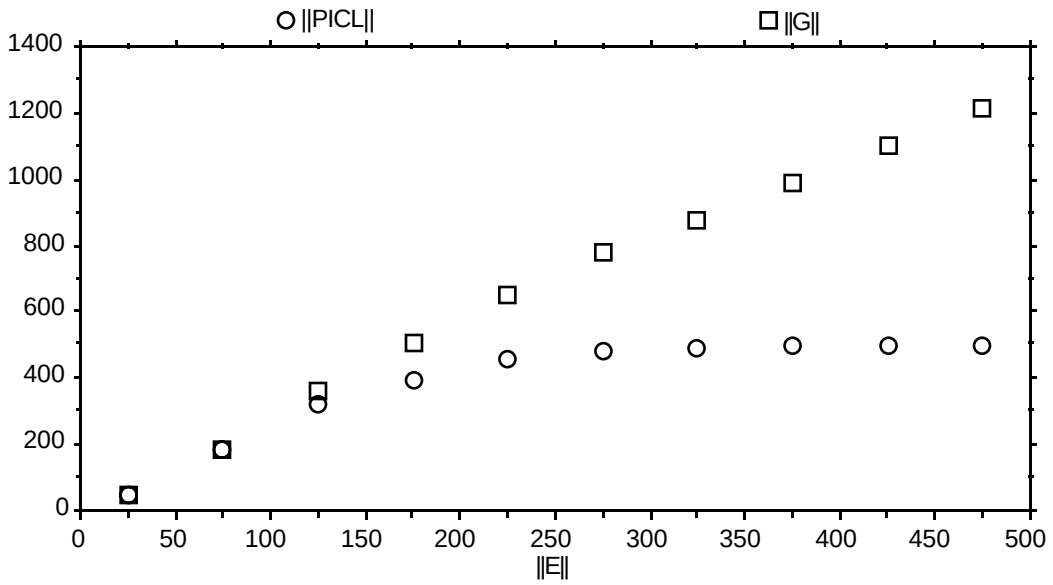

Figure 17. Number of elements for simulation with ||E'|| =500 and k=6.

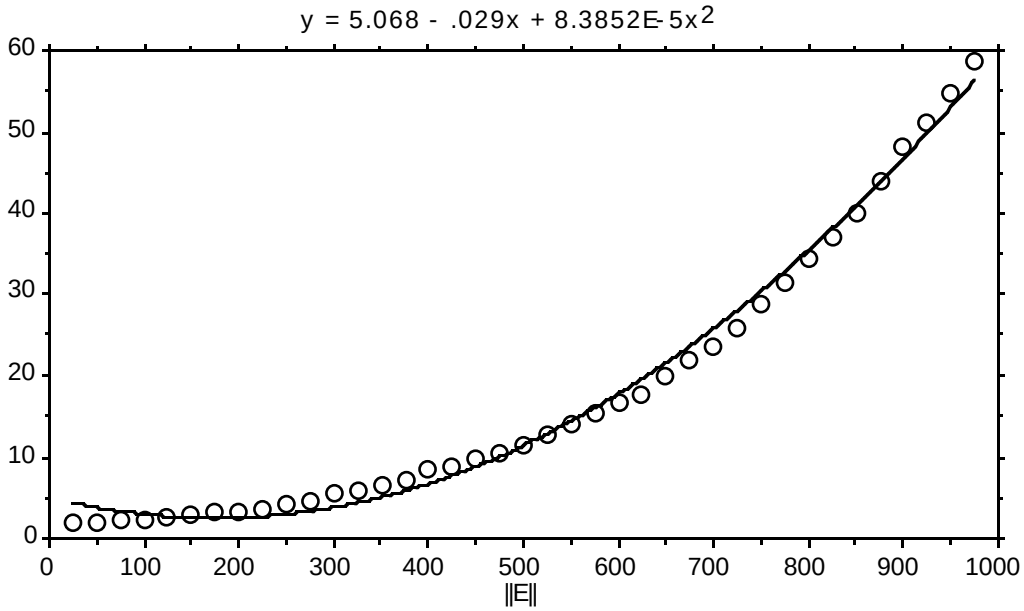

Figure 18. Regression CPU time for the data elements application using the incremental PICL algorithm.

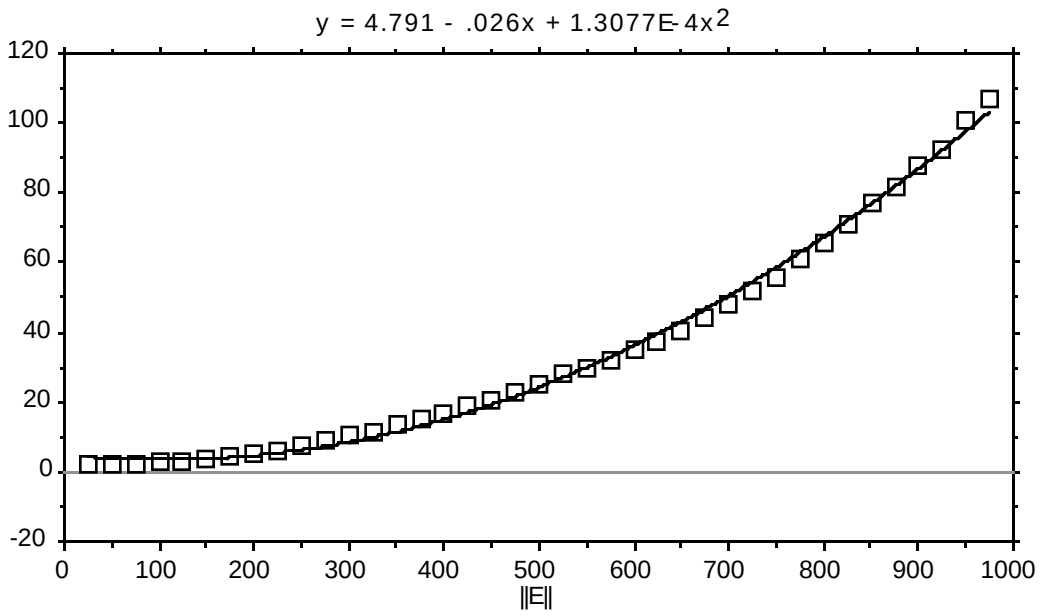

Figure 19. Regression CPU time for the data elements application using the concept lattice incremental algorithm.

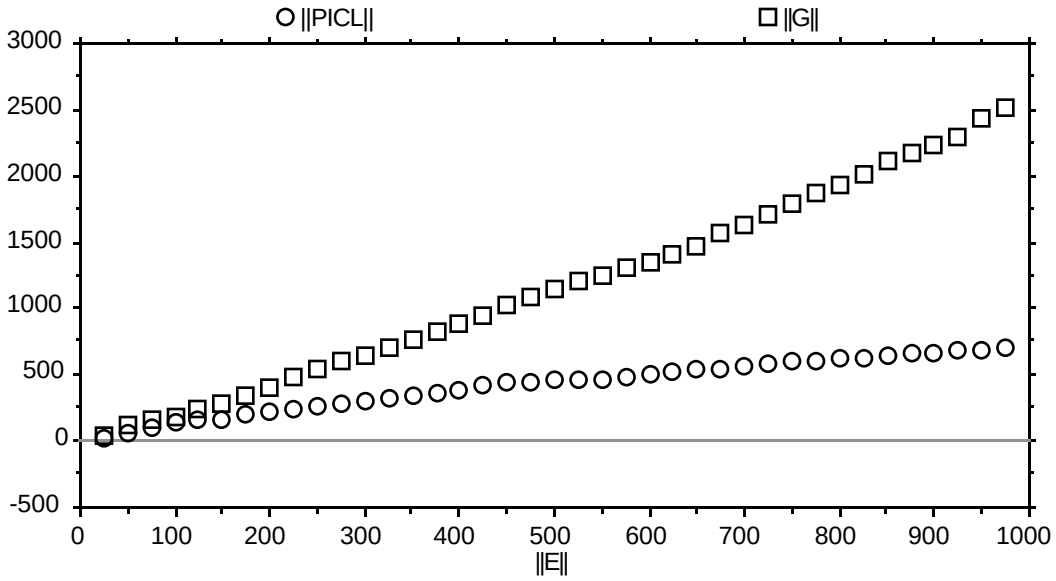

Figure 20. Number of elements for data elements application.

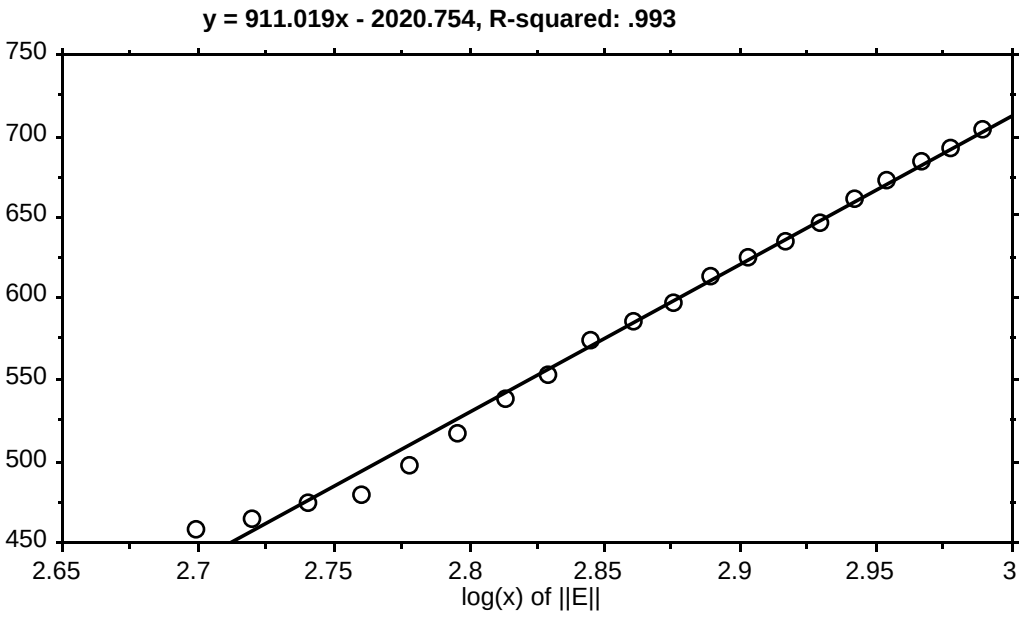

Figure 21. Regression for ||PICL|| versus log(||E||) for the data elements application.

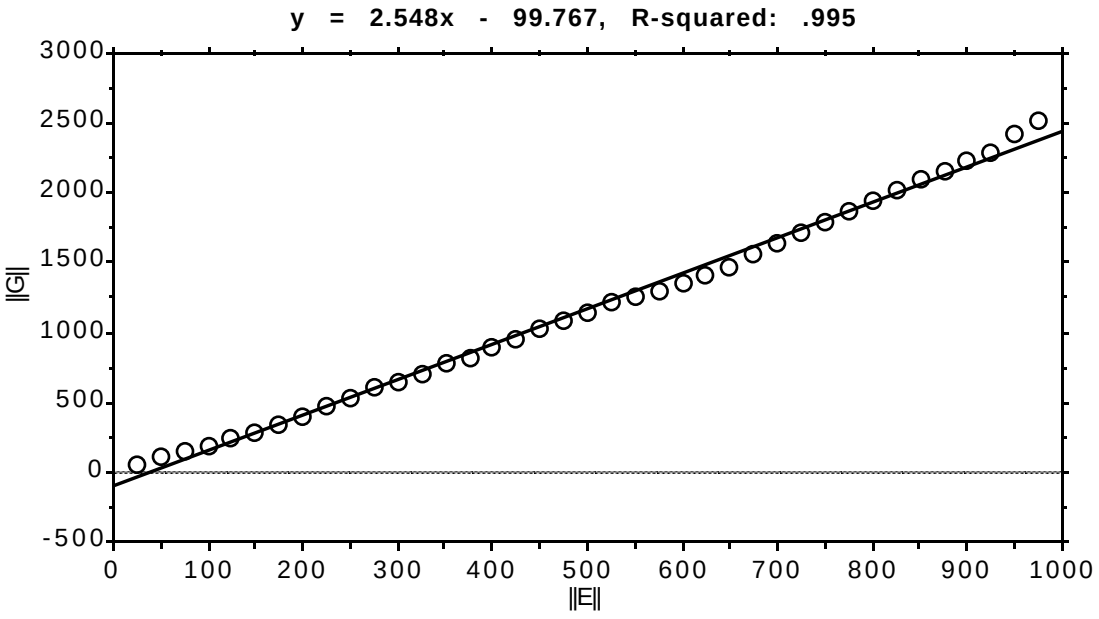

Figure 22. Regression for ||G|| versus ||E|| for the data elements application.

FIGURE CAPTIONS

Figure 1. Matrix representation of a binary relation R.
Figure 2. Concept lattice for the relation in Figure 1.
Figure 3. X-inheritance concept lattice.
Figure 4. X'-inheritance concept lattice.
Figure 5. Inheritance concept lattice.
Figure 6. PICL for the relation in Figure 1.
Figure 7. Initial screen for concept lattice or PICL.
Figure 8. Result of selection of "DATE" from Figure 7 for concept lattice.
Figure 9. Result after selection of "AAAAMMJJ" then "EFFECTIVE" from Figure 8.
Figure 10. Result of selection of "DATE" from Figure 7 for PICL.
Figure 11. Result after selection of "AAAAMMJJ" then "EFFECTIVE" from Figure 10 and display of list of data elements for current couple.
Figure 12. Sample screen from PICL generated using the methodology text.
Figure 13. Part of concept lattice for entity-attribute relationship.
Figure 14. Part of PICL for entity-attribute relationship.
Figure 15. Regression for CPU time with ||E'|| =500 and k=6 using the incremental PICL algorithm.
Figure 16. Regression for CPU time with ||E'|| =500 and k=6 using the incremental concept lattice algorithm.
Figure 17. Number of elements for simulation with ||E'|| =500 and k=6.
Figure 18. Regression CPU time for the data elements application using the incremental PICL algorithm.
Figure 19. Regression CPU time for the data elements application using the concept lattice incremental algorithm.
Figure 20. Number of elements for data elements application.
Figure 21. Regression for ||PICL|| versus log(||E||) for the data elements application.
Figure 22. Regression for ||G|| versus ||E|| for the data elements application.

ACKNOWLEDGMENTS

This research has been partly supported by NSERC (the Natural Sciences and Engineering Research Council of Canada), FCAR Funds (Fonds pour la Formation de Chercheurs et l'Aide à la Recherche) and the software engineering IT Macroscope Project, managed by the DMR Group Inc., and involving the "Centre de Recherche en Informatique de Montréal".

REFERENCES

Barbut, M. & Monjardet, B. (1970). *Ordre et Classification. Algèbre et Combinatoire, Tome II*. Hachette.

Basili, V. R., Caldiera, G. & Cantone, G. (1992). A Reference Architecture for the Component Factory. *ACM Transactions on Software Engineering and Methodology*, **1**(1), 53-80.

Berzins, V. & Luqi (1991). *Software Engineering with Abstractions*. Addison-Wesley.

Brookes, B. C. (1977). Theory of the Bradford Law. *Journal of Documentation*, **33**(3), 180-209.

Carpineto, C. & Romano, G. (1993). GALOIS: An Order-Theoretic Approach to Conceptual Clustering. In *Proceedings of the Machine Learning Conference*, pp. 33-40.

Carrington, D., Duke, D., Hayes, I. & Welsh, J. (1993). Deriving Modular Designs from Formal Specifications. *Software Engineering Notes (First ACM SIGSOFT Symposium on the Foundations of Software Engineering)*, **18**(5), 89-98.

Castano, S. & Antonellis, V. d. (1993). A Constructive Approach to Reuse of Conceptual Components. In *Proceedings of the Advances in Software Reuse, Selected Papers from the Second International Workshop on Software Reusability*, Lucca, Italy: IEEE Computer Society Press, pp. 19-25.

Devandu, P., Brachman, R. J., Selfridge, P. & Ballard, B. (1991). LaSSIE: a Knowledge-Based Software Information System. *CACM*, **34**(5), 35-49.

Fagan, J. L. (1989). The Effectiveness of a Nonsyntactic Approach to Automatic Phrase Indexing for Document Retrieval. *JASIS*, **40**(2), 115-132.

Faraj, N., Godin, R., Missaoui, R., David, S. & Plante, P. (1994). Évaluation de la contribution des termes composés syntaxiques pour l'indexation automatique. In *Proceedings of the 22e Congrès Annuel de l'Association Canadienne des Sciences de l'Information*, Montréal: pp. 340-360.

Fisher, D. (1987). Knowledge Acquisition via Incremental Conceptual Clustering. *Machine Learning*, **2**, 139-172.

Freeman, P. (1987). Reusable Software Engineering: Concepts and Research Directions. In P. Freeman (Eds.), *Tutorial: Software Reusability*, (pp. 10-23).

Gennari, J. H., Langley, P. & Fisher, D. (1990). Models of Incremental Concept Formation. In J. Carbonell (Eds.), *Machine Learning: Paradigms and Methods*, (pp. 11-62). Amsterdam, The Netherlands: MIT Press.

Gibbs, S., Tsichritzis, D., Casais, E., Nierstrasz, O. & Pintado, X. (1990). Class Management for Software Communities. *Communications of the ACM*, **33**(9), 90-103.

Godin, R. & Mili, H. (1993). Building and Maintaining Analysis-Level Class Hierarchies Using Galois Lattices. In *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'93)*, A. Paepcke (Ed.), Washington, DC: ACM Press, pp. 394-410.

Godin, R., Missaoui, R. & Alaoui, H. (1991). Learning Algorithms Using a Galois Lattice Structure. In *Proceedings of the Third International Conference on Tools for Artificial Intelligence*, S. Lee,B. Wah,N. G. Bourbakis, & W. T. Tsai (Ed.), San Jose, Calif.: IEEE Computer Society Press, pp. 22-29.

Godin, R., Missaoui, R. & April, A. (1993). Experimental Comparison of Navigation in a Galois Lattice with Conventional Information Retrieval Methods. *International Journal of Man-Machine Studies*, **38**, 747-767.

Godin, R., Saunders, E. & Gecsei, J. (1986). Lattice Model of Browsable Data Spaces. *Information Sciences*, **40**, 89-116.

Hutchens, D. H. & Basili, V. R. (1993). System Structure Analysis: Clustering with Data Bindings. In M. Shepperd (Eds.), *Software Engineering Metrics I: Measures and Validation*, Berkshire, England: McGraw-Hill.

Jones, T. C. (1984). Reusability in Programming: A Survey of the State of the Art. *IEEE Transactions on Software Engineering*, **10**(5), 488-494.

Krueger, C. W. (1992). Software Reuse. *ACM Computing Surveys*, **24**(2), 131-184.

Lebowitz, M. (1987). Experiments with Incremental Concept Formation: UNIMEM. *Machine Learning*, **2**, 103-138.

Maarek, Y. S., Berry, D. M. & Kaiser, G. E. (1991). An Information Retrieval Approach for Automatically Constructing Software Libraries. *IEEE Transactions on Software Engineering*, **17**(8), 800-813.

Mineau, G., Godin, R. & Missaoui, R. (1993). Induction of Generic Data Models by Conceptual Clustering. In *Proceedings of the The Fifth International Conference on Software Engineering & Knowledge Engineering*, San Francisco: pp. 554-564.

Mineau, G. W. & Godin, R. (1994). Automatic Structuring of Knowledge Bases by Conceptual Clustering. *IEEE Transactions on Knowledge and Data Engineering*, To appear.

Oosthuizen, G. D., Bekker, C. & Avenant, C. (1992). Managing Classes in Very Large Class Repositories. In *Proceedings of the Tools*, Paris: pp. 625-633.

Salton, G. & Buckley, C. (1988). Term weighting approaches in automatic text retrieval. *Information Processing and Management*, **24**(5), 513-523.

Wilde, N. & Huitt, R. (1992). Maintenance Support for Object-Oriented Programs. *IEEE Transactions on Software Engineering*, **18**(12), 1038-1044.

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.

Wille, R. (1992). Concept Lattices and Conceptual Knowledge Systems. *Computers Mth. Applic.*, **23**(6-9), 493-515.

Abstract 1
1. INTRODUCTION..... 2
2. CONCEPT FORMATION USING CONCEPT LATTICES AND VARIANT STRUCTURES 6
2.1 CONCEPT LATTICE 6
2.2 INHERITANCE CONCEPT LATTICE..... 8
2.3 PRUNED CONCEPT LATTICE..... 9
3. APPLICATIONS..... 11
3.1 RETRIEVAL..... 11
3.2 REUSE PACKAGING..... 14
4. COMPLEXITY OF THE METHODS 16
CONCLUSION..... 18
FIGURES 19
FIGURE CAPTIONS..... 41
ACKNOWLEDGMENTS 42
REFERENCES 43