JOURNAL OF SOFTWARE MAINTENANCE: RESEARCH AND PRACTICE
J. Softw. Maint: Res. Pract. 2002; **00**:1–32

A user-assisted approach to component clustering[†]

Kamran Sartipi[1,*,‡] and Kostas Kontogiannis[2,‡]

School of Computer Science[1]*, and*
Department of Electrical & Computer Engineering[2]*,*
University of Waterloo, Waterloo, ON, N2L-3G1, Canada

SUMMARY

In this paper, we present a user assisted clustering technique for software architecture recovery based on a proximity measure that we call component association. The component association measure is computed on the shared properties among groups of highly related system entities. In this approach, the software system is modeled as an attributed relational graph with the software constructs (entities) represented as nodes and data/control dependencies represented as edges. The application of data mining techniques on the system graph allows to generate a component graph where the edges are labeled by the association strength values among the components. An interactive partitioning technique and environment is used to partition a system into cohesive subsystems where the graph visualization aids and cluster quality evaluation metrics are applied to assess and fine tune the partition by the user.

KEY WORDS: partitioning; association; data mining; architecture recovery; similarity; graph

1. Introduction

Legacy software systems are mission critical systems that are operational approximately between 10 to 15 years [1]. Due to prolonged maintenance, such systems are difficult to maintain, evolve, or integrate and in most cases they deviate from their original design. In this context, architectural recovery is a key activity in supporting maintenance tasks such as re-engineering, objectification, or restructuring.

[*] Correspondence to: Kamran Sartipi, School of Computer Science, University of Waterloo, Waterloo, Ontario, N2L-3G1, Canada.
[‡] E-mail: {ksartipi,kostas}@swen.uwaterloo.ca
Contract/grant sponsor: IBM Canada Ltd. Laboratory - Center for Advanced Studies (Toronto) and the National Research Council of Canada.

Received 31 August 2001
Revised 19 September 2002

According to the related literature, the clustering-based approaches to software architecture recovery can be categorized into two groups. The first group encompass automatic or semi-automatic techniques [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12] and use a similarity metric (e.g., association coefficient, correlation coefficient, or probabilistic measures) which reflects a particular property among the system entities, and a clustering algorithm (e.g., agglomerative, optimization, graph-based, or construction) to partition the system into groups of related entities [10]. The approaches in the second group [13, 14] are based on tool usage, domain knowledge, and visualization means, to perform an iterative user-assisted clustering process. Such user-assisted techniques have been proven useful in handling large systems [13]. In this paper, we aim at a clustering technique that blends the advantages of both groups and provides means for partitioning a large monolithic software system into a collection of clusters as a possible architecture of the system.

In the approach presented in this paper, the software system is modeled as an *attributed relational graph* where the system entities are represented as nodes and data/control dependencies are represented as edges. The application of data mining techniques on this graph reveals associated groups of entities that possess a high-degree of data/control dependencies among the entities. Hence, these groups are suitable to form clusters based on a similarity metric, namely *entity-association*. This similarity metric encodes the structural property of the groups of entities that are related by *maximal association*. The maximal association property is defined in the form of a maximal set of entities that all share a maximal set of features. In a further step, the entity-association metric is used as a primitive to define the *component-association* metric that measures the degree to which the entities in one component are related to the entities in another component.

In our work, a component is defined to be either a *file*, or a *module* of system entities such as functions, datatypes, variables, or a *subsystem* that is a collection of system files. This allows us to represent a system as a *component graph*, where the nodes represent system files and the labeled edges represent component association values among the files. Such association values can be quantized and classified into four ranges (strong, medium, loose, and weak) each representing a different strength of association between two files. In this context, we propose an iterative partitioning technique and environment that emphasizes on pre-processing the raw system data to a level that either the tool or the user can perform the clustering operation. Visualization of the component graph allows the user to fine-tune the automatically generated system partition.

We have implemented a prototype reverse engineering toolkit (Alborz [15]) to recover the architecture of a software system in the form of components that share common features. The toolkit presents the results in the form of HTML pages to be browsed and graphs to be visualized, and provides modularity metrics to assess the quality of the software system and its partitioning into subsystems.

The contributions of this paper can be summarized as follows: i) providing a user-assisted clustering environment that can be applied on large systems; ii) proposing association-based similarity measures between two system entities and between two components based on data mining techniques; iii) presenting a new partitioning clustering technique based on a similarity threshold to control the quality of the partition.

This paper is organized as follows. The related work is discussed in section 2. Section 3 presents an overview of the partitioning environment. Section 4 discusses the adopted graph based system representation. Section 5 discusses software quality measure and extracting groups of entities with maximal association using data mining techniques. Sections 6 and 7 define the association metrics between two entities and between two components, respectively. Section 8 presents the algorithms for iterative partitioning technique. Section 9 discusses the case studies of six software systems and evaluates the proposed partitioning technique. Finally, section 10 concludes the paper and provides insights into the future research.

2. Related work

The closest clustering techniques to our approach in this paper pertain to the application of "*concept lattice analysis*". A concept is defined as a group of entities with maximal association and a concept-lattice is generated to view the structure of relations among entities in a small program. However, in medium or large software systems (+50 KLOC) the concept lattice becomes so complex that the visual characteristics of the lattice are obscured. In such cases, the engineers must seek automatic partitioning algorithms to assist them in finding distinct clusters of highly related concepts. Sif [16] uses a repairing technique by adding extra relations to make the generated concept lattice *well-formed* in order to provide easier partitioning. The main drawback of this approach is the large number of generated partitions that requires high user-involvement for reducing the number of partitions to a manageable set for investigation. Snelting [17] uses a technique called "horizontal decomposition" to partition a lattice of procedures and variables into modules. However, the overwhelming number of interferences between concepts in the lattice of a real system prevents such a horizontal partitioning. In comparison with concept lattice approaches, we define a similarity measure which encodes the structural characteristics of the neighboring concepts and uses this metric to cluster the groups of closely related concepts.

Mancoridis [8] proposes a method to partition a group of system files into a number of clusters using a hill-climbing search and neighboring partitions, where the initial partition is randomly selected. In comparison, our method computes a collection of rather separated singleton clusters as an initial partition and the iterative partition can then proceed on computing an optimal partition.

Tzerpos [18] uses a number of system structural properties as evidences to cluster the system files into a hierarchy of clusters. The method uses subgraph dominator nodes to find subsystems of almost 20 members, and builds up the hierarchy of subsystems accordingly. To simplify the computation, the interactions of more than 20 links to/from a file are disregarded. In contrast, our technique does not assume any pre-existing structure for the system such as directory structure, instead relies on overall data/control flow dependencies among the system entities to be used for clustering.

Ferneley [19] defines two sets of measures on the coupling and control flow analysis, each classified into three measures with increasing complexity of measurements. For intra-module control flow measures the author considers logical constructs such as selection/iteration, and their scopes as different levels of refinement for measurement. Also for inter-module coupling

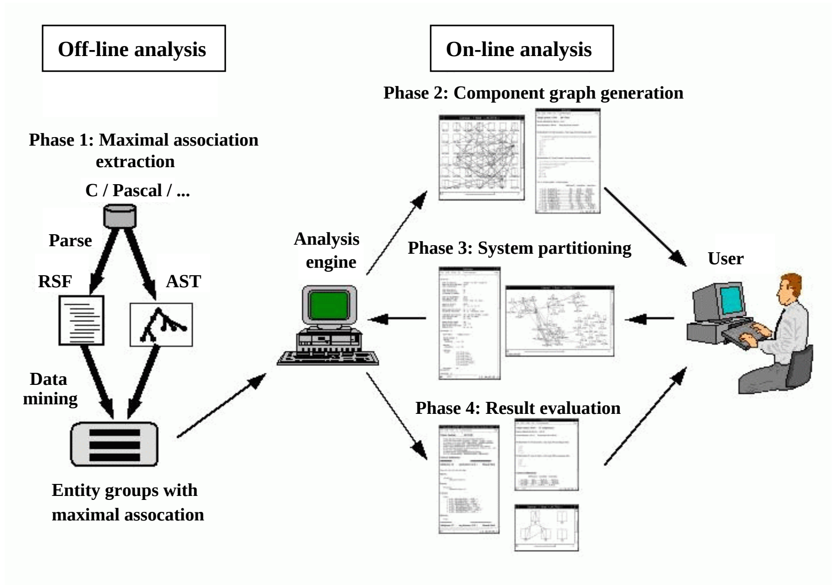

Off-line analysis
On-line analysis
Phase 1: Maximal association extraction
C / Pascal / ...
Parse
RSF
AST
Data mining
Entity groups with maximal assocation
Analysis engine
Phase 2: Component graph generation
Phase 3: System partitioning
Phase 4: Result evaluation
User

Figure 1. The environment for software system partitioning based on component association.

the author considers the structure of data being passed, and single/multiple input as levels of refinement to be considered. In our approach we measure the quality of the partitioning technique by considering control-flows at the function level not logical constructs, and for data-flow related measurements we consider structured data and multiple inputs among the clusters.

Finally, Lakhotia [20] provides a unified framework that categorizes the different software clustering techniques and translates them into the framework notation to be compared with.

3. Overview of the user-assisted partitioning environment

Figure 1 illustrates the proposed environment for association-based system partitioning. The partitioning environment consists of four phases classified into two parts namely *off-line* and *on-line* analyses.

Phase 1: Maximal association extraction. The software system (i.e., a C program) is parsed and a database of entities and relationships in the form of abstract syntax tree (AST) or textual formats is generated. The entity-relationship database is represented as an *attributed relational graph* [21], namely the system-graph denoted as $G = (N, R)$. The application of data mining algorithms on this graph extracts the groups of entities with high data/control flow dependencies. These groups of entities are related with maximal association. This phase may take several hours to complete for a large system.

Phase 2: Component graph generation. The group of entities with maximal association are analyzed to compute the association among the system files and generate a *component graph* $G^C = (N^C, R^C)$. The association links between the files in G^C are then quantized into value ranges in order to be color-coded and visualized by graph visualization tools. In the resulting quantized component graph the nodes represent system files and the edges represent the in-between association strengths to be used for partitioning process.

Phase 3: System partitioning. Based on an iterative partitioning algorithm to be discussed in this paper, the component graph G^C is partitioned into clusters (i.e., subsystems of files) either automatically by tool, or manually by visualizing and manipulating the quantized graph G^C with color-coded edges.

Phase 4: Result evaluation. The tool analyzes the partitioned system and provides quality evaluation of the clustered subsystems in the forms of: closeness value for each file in a subsystem to the other files of subsystem; modularity quality of the partition; and the graph of the clustered subsystems. The user investigates the evaluation results and, if needed, modifies the obtained clusters to incorporate the domain knowledge and system documentation, and consequently repeats the phases 3 and 4 until the partitioning meets specific user criteria.

4. Graph based system representation

In a software system, the entities can be specified according to a domain model for the corresponding programming language. These entities are instantiations of the domain model constructs such as: function, datatype, statement, assignment, variable, file. In clustering analysis, the granularity level of the selected source code entities depends on the purpose of the analysis. For example, function, datatype, and variable are used for clustering at the module level, and file is used for clustering at the system level.

Figure 2 illustrates the mappings from the entities and relationships in the domain of a typical procedural language onto the entities and relationships in the domain that is suitable for architectural level analysis. The entities at the architectural level constitute a subset of the whole entities in the software system. For example, the entities such as *local variables* and *scalar-types* are deleted at the architectural level. Each relation at the architectural level is an aggregation of one or more relations at the software system level. For example the operation "function *foo* references or updates global-variable *kam*" is abstracted as "F_j *use-V* V_m", where F_j and V_m are unique identifiers for "function *foo*" and "global-variable *kam*", and *use-V* is an aggregation of two relations "reference and update". Each abstract entity or abstract relation has three attributes *label, type*, and *location*, as defined below.

- *label:* for entities the label denotes: i) a full path-name as a unique name for each entity in the software system; ii) a unique identifier to refer to an entity, e.g., F4, L6, T32; for relations the label denotes a pair of source and sink entities that are related.

Entities	
Software System level	**Architectural level**
source-file "*main.c*"	abstract-file L_i
function "*foo*"	abstract-function F_j
aggregate-type "*bar*" or array-type "*bar*"	abstract-type T_k
global-variable "*kam*"	abstract-variable V_m

Relationships	
Software System level	**Architectural level**
function "*foo*" calls function "*foobar*"	F_j *use-F* F_x
function "*foo*" passes, receives, or uses aggregate-type / array-type "*bar*"	F_j *use-T* T_k
function "*foo*" references or updates global-variable "*kam*"	F_j *use-V* V_m
source-file "*main.c*" defines function "*foo*", defines aggregate-type / array-type "*bar*", defines global-variable "*kam*"	L_i *cont-R* F_j L_i *cont-R* T_k L_i *cont-R* V_m

Figure 2. Domain model used in this paper. Entities at the architectural level are a subset of the software system entities, where L_i, F_j, T_k, V_m are unique entity identifiers. Each relationship at the architectural level is an aggregation of one or more relationships in the software system.

- *type:* L, F, T, V are the types of entities for abstract-file, abstract-function, abstract-type, and abstract-variable, respectively. *use-F, use-T, use-V, cont-R* are the types for relations.
- *location:* denotes the *source file number* and *line number in file* where the entity or the relation between two entities are defined in the software system.

Attributed relational graph

In this section, we briefly introduce the underlying concept of *Attributed Relational Graph* (ARG) that we use to represent a software system entities and relationships, based on the notation presented in [21].

At the architectural level the Attributed Relational Graph of the software system is denoted as the *system-graph* which is defined as a six-tuple $G = (N, R, A, E, \mu, \epsilon)$ (or simply $G = (N, R)$), where the nodes are entities and the edges are relationships defined at the architectural level in Figure 2, and the attributes for nodes and edges are defined above.

A system-graph $G = (N, R, A, E, \mu, \epsilon)$ is defined as:

- N : $\{n_1, n_2, ..., n_n\}$ is the set of attributed nodes, or entities at the architectural level.

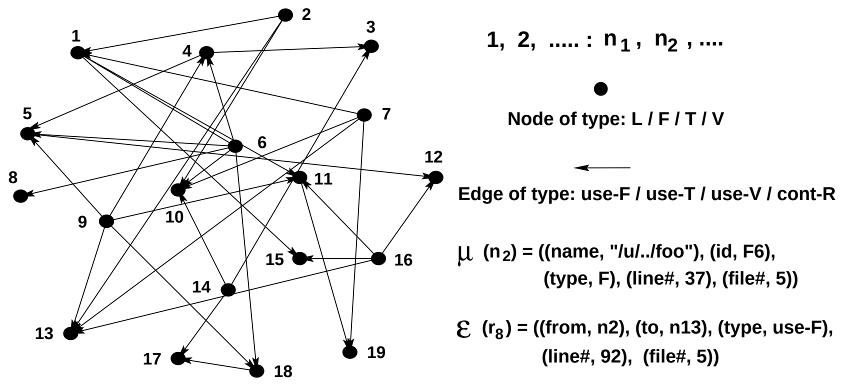

1, 2, ... : n1 , n2 , ...
Node of type: L / F / T / V
Edge of type: use-F / use-T / use-V / cont-R
μ (n2) = ((name, "/u/../foo"), (id, F6), (type, F), (line#, 37), (file#, 5))
ε (r8) = ((from, n2), (to, n13), (type, use-F), (line#, 92), (file#, 5))

Figure 3. An Attributed Relational Graph representation of a system-graph $G = (N, R, A, E, \mu, \epsilon)$.

- R : $\{r_1, r_2, ..., r_m\}$ is the set of attributed edges, or relationships at the architectural level.
- A : alphabet for node attribute values such as node labels and node types.
- E : alphabet for edge attribute values such as edge labels and edge types.
- $\mu : N \rightarrow (A \times A)^p$: a function for returning "node attribute, node attribute value" pairs where p is a constant.
- $\epsilon : R \rightarrow (E \times E)^q$: a function for returning "edge attribute, edge attribute value" pairs where q is a constant.

Figure 3 illustrates the system-graph of a small system with 19 nodes. In the system-graph G, examples of node and edge labeling functions μ and ϵ are as follows:

- $\mu(n_2)$ = ((name, "/u/.../foo"), (id, F6), (type, F), (line#, 37), (file#, 5)) indicating that node n_2 of the system-graph G is of type abstract-function with name *"/u/.../foo"* and identifier F6 which has been defined in line 37 of the source file 5; and
- $\epsilon(r_8) = ((from, n_2), (to, n_{13}), (type, use\text{-}F), (line\#, 92), (file\#, 5))$ indicating that the edge r_8 is of type *use-F*, i.e., the software system function represented by the node n_2 calls the software system function represented by the node n_{13}; and the function-call occurs in line 92 of file 5.

In the following sections, we apply data mining techniques on the system-graph G to extract groups of maximally related entities.

5. Software quality measure

Coupling and *cohesion* are two major metrics for assessing the quality of a software system in terms of understandability and maintainability. In the early 1970s, researchers and practitioners noticed that software designers collect certain program parts into the same module according to particular relationships between a set of actions they perform. These relationships were organized by Stevens, Myers, Constantine, and Yourdon [22, 23] as the levels of coupling and cohesion among or within the software systems' modules, which are now considered as the standard software quality measures [24].

In a software system consisting of modules, *coupling* is a "measure of the relative interdependence among the modules" [24] and is measured by seven ordinal levels from the weakest to strongest as *no-coupling, data, stamp, control, external, common*, and *content. Cohesion* is a "measure of the relative functional strength of a module" [24] and is measured by seven ordinal levels from weakest to strongest as *coincidental, logical, temporal, procedural, communicational, sequential*, and *functional.* A software system whose components demonstrate high-cohesion and low-coupling is known as a modular system.

Cohesion (functional strength) is difficult to measure because most of the functions in a module are composed of smaller functions, hence require to investigate a function hierarchy for cohesion measure [25]. The functional strength must be interpreted by the software engineer, hence cohesion is a subjective measure [26]. Recently, the researchers [25, 26, 27, 28] attempted to provide objective measures that closely relate to the original seven levels of cohesion proposed by Stevens *et al.*

Chapin [25] proposes an objective method to appraise the coupling and strength (cohesion) of a software system component. The approach uses message tables and two decision tables that use questions to direct the analyst or programmer to the appropriate level of the coupling or cohesion. The use of these aids makes the appraisal process more objective and practical than the traditional ones. Lakhotia [26] defines a number of rules of logic using the data/control dependencies that translate the seven cohesion levels into formal description for a module to be evaluated. Bieman [28] proposes a sophisticated intra-module cohesion measure based on data slices to determine the extend to which a module approaches the ideal of functional cohesion. In [27] an experimental evaluation of a group of graduate students (with the knowledge about cohesion) was conducted to study whether or not the Stevens *et al.* rules can be used to determine the cohesion of a module from its source code. The overall conclusion drawn was that the cohesion levels are not so intuitively obvious to be used reliably. In some cases the students confused the highest level of cohesion with the lowest level.

Misic [29] noted that the basic concepts of coupling have never been challenged and cohesion can also be expressed in terms of coupling, such that cohesion be viewed as a close relative of internal coupling, or a variation of it. Patel [30] uses a vector of counters for variables of each program (function) and when the program accesses a variable the counter corresponding to that variable in the vector is incremented. Consequently the cohesion between the programs are calculated based on the number of shared variables and the counter for that variable. Mancoridis [8] defines a modularity metric for a software system based on inter-/intra-module connectivity. Lindig [17] defines the cohesion of a module in terms of sharing variables by the module's procedures. These authors view the cohesion of a module as a measure of "*coherency*" [29], "*sharing*" [30, 17], or "*intra-connectivity*" [8] among the functions, which is considered as a form of *external* property of the system functions. In this context, a number of common attribute values among the functions can determine the cohesion as the degree of sharing different sets of: global variable reference, function call, or data type usage.

The property of sharing common attributes is known as the *association similarity* metric in the clustering literature [10] and are widely used in producing cohesive clusters. Two common association-based similarity metrics are the *Jaccard* and *matching coefficient* metrics [10]. These metrics measure the size ratio of different weighted unions/intersections of the attribute sets of two functions. However, if groups of more than two functions are considered then a new sharing property for the group must be defined. In this form, the maximum number of shared attribute values among the group, known as *maximal association*, is an interesting property for defining a similarity measure among a collection of system entities, which is proposed in this paper.

5.1. Maximal association

Informally, maximal association is defined in a group of entities in the form of a maximal set of entities that all share the same relations to every member of another maximal set of entities.

For every set of functions, denoted as $\mathcal{F}$, we can determine a set of shared entities , denoted as $\mathcal{E}$, where every function f in $\mathcal{F}$ has a relation *rel* to an entity e in $\mathcal{E}$. For example, two functions f and g can share the datatype t and variable v by the relations *use-T* and *use-V*, respectively.

The operation *sh-ents*($\mathcal{F}$) returns the set of shared entities $\mathcal{E}$ for the set $\mathcal{F}$ as follows:

$$\textit{sh-ents}(\mathcal{F}) = \{e \mid \forall f \in \mathcal{F};\ \exists rel : X \bullet X \in \{\textit{use-F, use-T, use-V}\} \wedge (f,\ e) \in rel\}. \quad (1)$$

Similarly, for every set $\mathcal{E}$ of entities we can determine a set of functions $\mathcal{F}$, where every function f in $\mathcal{F}$ has a relation *rel* to an entity e in $\mathcal{E}$. The operation *sh-funcs*($\mathcal{E}$) returns the set of sharing functions $\mathcal{F}$ for the set $\mathcal{E}$ as follows:

$$\textit{sh-funcs}(\mathcal{E}) = \{f \mid \forall e \in \mathcal{E};\ \exists rel : X \bullet X \in \{\textit{use-F, use-T, use-V}\} \wedge (f,\ e) \in rel\}. \quad (2)$$

A set of functions $\mathcal{F}$ and a set of entities $\mathcal{E}$ are related by maximal association, iff:

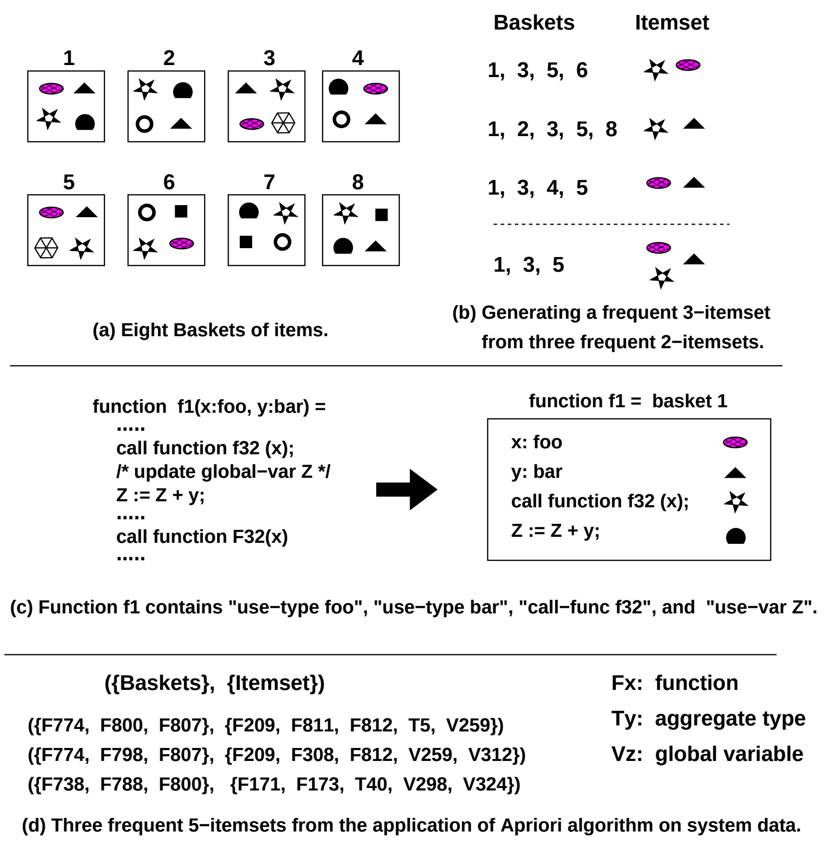
$$\mathcal{F} = \textit{sh-funcs}(\mathcal{E}) \quad \wedge \quad \mathcal{E} = \textit{sh-ents}(\mathcal{F}). \quad (3)$$

In this form, no larger set of functions $\mathcal{F}'$ ($\mathcal{F}' \supset \mathcal{F}$) exists such that $\mathcal{F}'$ and the set of entities $\mathcal{E}$ are related by maximal association. Similarly, no larger set of entities $\mathcal{E}'$ ($\mathcal{E}' \supset \mathcal{E}$) exists such that $\mathcal{F}$ and $\mathcal{E}'$ are related by maximal association.

In the following, the application of the data mining algorithm *Apriori* [31] in detecting maximal association among entities is discussed.

5.2. Data mining

Data mining or Knowledge Discovery in Databases (KDD) refers to a collection of algorithms for discovering or verifying interesting and non-trivial relations among data in a large database [32]. A substantial body of data mining literature is based on extensions of the *Apriori algorithm* by Agrawal [31], and relate to the concept of *market baskets* and their *items* in databases. A *k-itemset* is a set of items with cardinality $k > 0$. A *frequent itemset* is an itemset whose elements are contained in every basket of a group of baskets (namely *supporting baskets*).


Baskets
Itemset
1 2 3 4
5 6 7 8
1, 3, 5, 6
1, 2, 3, 5, 8
1, 3, 4, 5
1, 3, 5
(a) Eight Baskets of items.
(b) Generating a frequent 3−itemset from three frequent 2−itemsets.
function f1 = basket 1
x: foo
y: bar
call function f32 (x);
Z := Z + y;
(c) Function f1 contains "use−type foo", "use−type bar", "call−func f32", and "use−var Z".
Fx: function
Ty: aggregate type
Vz: global variable
(d) Three frequent 5−itemsets from the application of Apriori algorithm on system data.

```
function  f1(x:foo, y:bar) =
   .....
   call function f32 (x);
   /* update global−var Z */
   Z := Z + y;
   .....
   call function F32(x)
   .....
```

({Baskets},	{Itemset})
({F774, F800, F807},	{F209, F811, F812, T5, V259})
({F774, F798, F807},	{F209, F308, F812, V259, V312})
({F738, F788, F800},	{F171, F173, T40, V298, V324})

Figure 4. (a),(b) The notion of database baskets and frequent itemsets. (c) The mapping of the entities and relationships in a software system onto the baskets and items in data mining. (d) Representation of the frequent itemsets in the system.

The cardinality of this group of baskets must be greater than a user-defined threshold called *minimum-support*. The frequent itemsets are generated by the *Apriori* algorithm [31] that first generates the groups of frequent itemsets and then extracts the association rules in the form of 40% of baskets that contain the set of items X also contain the set of items Y.

Figure 4 illustrates the application of the Apriori algorithm in reverse engineering. In Figure 4(a), the market baskets and different kinds of items inside the baskets are shown, where each element represents all items of the same kind in a basket. Figure 4(b) demonstrates one iteration of the iterative generation of the frequent itemsets using the Apriori algorithm. The frequent i-itemsets are computed from the frequent (i-1)-itemsets obtained in the previous iteration. In each iteration i, the algorithm produces all frequent itemsets in the form of tuples $(\{baskets\}, \{items\})$ such that:

$$\{baskets\} = sh\text{-}funcs(\{items\}) \quad \wedge \quad \{items\} = sh\text{-}ents(\{baskets\}). \tag{4}$$

Hence, the functions as baskets and the entities (i.e., functions, datatypes, variables) as items are related by maximal association.

At the top part of Figure 4(b), three frequent 2-itemsets along with their container baskets are shown, from which the algorithm generates a frequent 3-itemset. In this example, the minimum-support (i.e., the minimum number of baskets) can be 3 or less. The resulting frequent 3-itemset exists in all basket 1, 3, and 5.

Figure 4(c), demonstrates the mapping from a function definition in a software system onto the notion of basket and items in the data mining domain. In our approach, a basket is a file or a function and the basket items are the system functions, datatypes, and global variables that are called or used according to the domain model represented in Figure 2. Figure 4(d) represents a small portion of frequent 5-itemsets extracted from a software system. The first line is interpreted as: all the functions F774, F800, F807 call functions F209, F811, F812, and use aggregate type T5 and global variable V259. The Apriori algorithm generates all the frequent itemsets and stores them into large groups based on the size of itemsets. The similarity measure between two system entities are extracted by scanning the stored frequent itemsets, which is discussed in the next section.

6. Association measure between two entities

In this section, we define *entity association* between two system entities based on the notion of association in a graph.

Association in a group of graph nodes is a property, where two or more source nodes share one or more sink nodes (through direct graph edges). A *source node* is a node where an edge emanates from it. A *sink node* is a node where an edge points to it. In analogy with data mining terminology, we refer to the source nodes as the "*basketset*" and the sink nodes as the "*itemset*". In this sense, the whole group of itemset and basketset are denoted as an *associated group*.

The entity association between two system entities e_i and e_j, denoted as $entAssoc(e_i, e_j)$, is defined as the *maximum* of the association value between e_i and e_j, considering that e_i and e_j may belong to more than one associated group g_x with a different association value in each group g_x. Formally:

$$entAssoc(e_i, e_j) = max_{g_x} \ (|itemset(g_x)| + w * |basketset(g_x)|) \tag{5}$$

where, $0 < w < 1$ is the weight of the sharing entities compared with the shared entities and is discussed later. The entity association is considered as a measure of similarity between two entities in a software system and allows to:

- identify the members of a group of highly related entities in a system.
- consider the datatypes and variables as members of a group including functions, as opposed to considering them as attribute-values of functions which cause only the functions to be grouped.

In general, the number of shared entities (items) contributes more on the closeness of the entities than the number of sharing entities (baskets), if a group of entities are examined for their similarity. We justify this property using a social analogy to software systems:

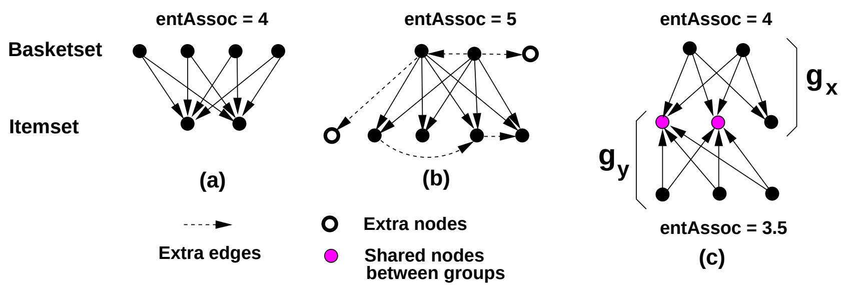


Figure 5. Illustrating the notion of *entity association* as a similarity measure between two entities.

"*Consider 10 people that eat in the same restaurant and go to the same library. These people can be friends or not. If the number of these people increases from 10 to 20 it does not necessarily increase the level of mutual friendship among them. Now consider the same 10 people and increase the number of their commonalities. For example, suppose they also live in the same building and go to the same club. These people have high likelihood to be friends, since a high number of shared interests is most often an indication of a high level of friendship among people.*"

The lower values of w (close to 0) cause that the $entAssoc(e_i, e_j)$ be insensitive to the number of sharing entities in an associated group, and vice versa. Based on the empirical results and the above mentioned property, we use a value of $w = 0.5$. The value of $entAssoc(e_i, e_j)$ is a positive *real* number which is not normalized since it measures a property in a single group of entities, not between two groups of entities which allows to normalize the metric. Hence, its value is not restricted between 0 and 1, instead it depends on the size and form of the group of entities in g_x. A possible way for normalization is to find the *maximum* of the association values in the system and divide all other values to it.

Figure 5 illustrates the notion of entity association similarity metric using four associated groups. In Figure 5(b) the extra nodes and edges that may exist among the nodes and edges of an association group are shown. However, only the solid nodes are the members of the associated group and extra edges do not affect the association value. Figure 5(c) illustrates two associated groups g_x and g_y with shared nodes. The grey-color nodes are the members of both groups with different association values. In such cases, the association value of a node is inherited from the group with larger association value. The entity association is considered as a measure of similarity between two entities in a software system.

7. Association measure between two components

In this section, we define *component association*, denoted as $compAssoc(C_i, C_j)$, between two system components C_i and C_j based on the similarity between two entities ($entAssoc$) in a graph.

A system *component* is a named grouping of the system entities such as files, functions, aggregate types, and global variables. We say that a component "contains" the entities it defines and each system entity can be contained in only one component. Furthermore, a component interacts with other components through *importing* and *exporting* of simple entities such as functions, aggregate types and global variables.

A component C_i is a member of a disjoint set of components $\{C_1, .., C_l\}$ that constitute a partitioning $P(N)$ of the system entities N according to a particular relation $\mathcal{R}$ among the system entities N, e.g., association relation.

A component C_i consists of three parts: i) *contains* part, denoting a set of system entities that are defined in the component C_i; ii) *imports* part, denoting entities that are used by the component C_i but are contained in another component C_j; and iii) *exports* part, denoting entities that are contained in component C_i and are used by other components.

If a *component* C_i contains a file L_k and file L_k contains a simple entity F_m then *component* C_i also contains the simple entity F_m. Therefore the containment relation is transitive.

We consider two kinds of components in a software system, denoted as *module* and *subsystem*, according to the type of system entities they contain and interact. A *module* M_i is a *component* C_i that *contains* simple entities (functions, datatypes, variables), and *imports* and *exports* simple entities. Therefore, a file can also be considered as a module and be treated as a component.

A *subsystem* S_i is a *component* C_i that *contains* composite entities (files) as well as their contained simple entities (functions, datatypes, variables), and *imports* and *exports* simple entities. In this paper, a component is a subsystem.

The component association *compAssoc*(C_i, C_j) is computed as the average of similarities between all pairs of entities that are made up of one entity from each component, as follows:

$$compAssoc(C_i, C_j) = \frac{\sum_{k=1}^{|C_{i_{contains}}|} \sum_{m=1}^{|C_{j_{contains}}|} entAssoc(n_k, n_m)}{|C_{j_{contains}}|}. \qquad (6)$$

In equation (6), the first summation iterates over every entity in component C_i and the second summation iterates over every entity in component C_j in order to add the similarity values $entAssoc(n_k, n_m)$ between every pair of entities, one entity in each group, i.e., n_k is in component C_i and n_m is in component C_j. The term $|C_{j_{contains}}|$ denotes the cardinality of component C_j. This equation is not symmetric with respect to the components C_i and C_j, i.e., $compAssoc(C_i, C_j) \neq compAssoc(C_j, C_i)$. The unit for $compAssoc(C_i, C_j)$ is *"association value per entity"*.

We define the notion of *component graph* which is central to the proposed partitioning technique. The *component graph* $G^C = (N^C, R^C)$ is defined using the component association values, as:

$$N^C = \{C_i \mid C_i \textit{ is a component}\}$$
$$R^C = \{e_k \mid e_k = (C_i, C_j) \textit{ is a component association link}\} \qquad (7)$$

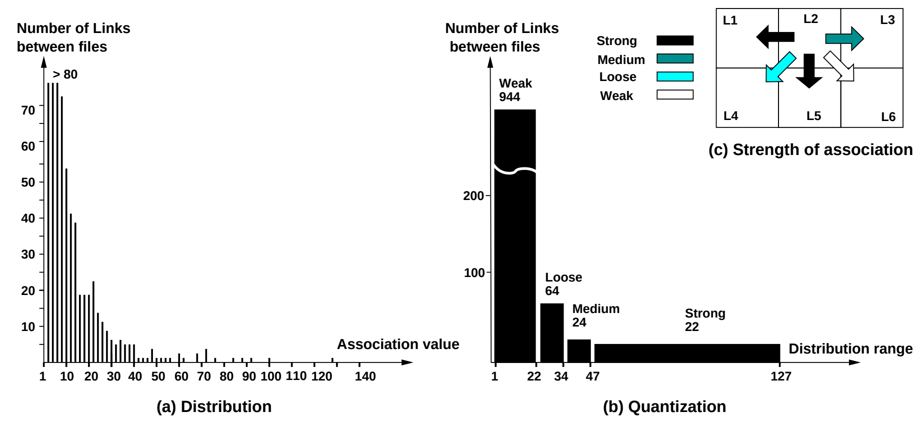


Figure 6. (a) Distribution and (b) Quantization of the component association values in the Clips system. (c) Strength of association between file L2 and other files in a system of six files.

where, the component association link e_k is annotated with the value of the $compAssoc(C_i, C_j)$.

7.1. Component association quantization

In a component graph G^C the values for component association are distributed over a broad range that is not suitable for graph visualization. In order to allow a user/tool cooperative partitioning process based on graph visualization, a quantization method is used to classify the values of component association into four ranges *strong, medium, loose*, and *weak* (denoted as *strength of the association*). Each range can be color-coded to be viewed and interpreted in a graph visualization tool. These ranges are defined below:

- *Strong association*: an indication of significant interaction among the entities of two components. Separate groups of components where each group of component has internal links with strong association, are proper candidates as the cores of distinct subsystems in a system decomposition. The functionality of these core components can be investigated to assign meaningful names for the subsystems. A strong self association of a component C_i, i.e., $compAssoc(C_i, C_i)$, indicates a high level of relationship among the entities of C_i.
- *Medium association*: an indication of high to medium interaction between two components. This type of association is considered for collecting the components around the core of a subsystem.
- *Loose association*: an indication of low interaction between two components. This type of association is used for grouping the yet ungrouped components, similar to the concept of *orphan adoption* [33]. The loose association may also be used for finding the commonly used components in a system.

- *Weak association*: an indication of insignificant or purely coincidental interaction between two components. This type of component interaction can be ignored for all practical purposes.

The distribution of the *association-link quantity* versus *association value* is the basis for determining the range of each association strength. According to the experimentations with a number of systems in different domains (section 9.1), this distribution decreases very fast with the increase of the association value (almost an inverse exponential distribution). Figure 6(a) illustrates such distribution for the Clips system with 44 files.

In order to produce a four-range association strength diagram that traces the envelop of such an inverse exponential distribution, we define a heuristic *quantize*(G^C) that uses the following constraints on the relative numbers of links (link quantity) in the consecutive ranges.

- The quantities of the *strong* and *medium* associations are almost equal, with higher quantity for the *medium* association.
- The quantity of the *loose* associations is approximately three times higher than the quantity of the *medium* association.
- The rest of the association values constitute the *weak* associations.

The heuristic *quantize*(G^C) requires a system-dependent value for the quantity of an association range to start with. We set the quantity of the strong association links to a number between 50% to 60% of the number of the system files, which tends to produce good results.

This heuristic attempts to accommodate the distribution of the association values so that the closest numbers to the above approximated values are achieved, as illustrated in Figure 6(b). The affect of the quantization process is to map the component association values of the edges in the component graph G^C, from a broad range of values onto a number between 1 and 4.

Figure 6(c) demonstrates a graphical representation of the quantization of the association values in part (a). In this example, a system of six files is considered where the association links of file L2 on other system files is shown. File L2 has strong association on files L5 and L1, medium association on file L3, and low association on other files. The strength of association between file L2 and other files have been color-coded to be viewed and distinguished in a graph visualization tool.

8. Partitional clustering technique

In this section, an automated partitional clustering technique for subsystem recovery is discussed. In the context of software reverse engineering the clustering algorithms can be categorized as: i) *hierarchical algorithms*, where each entity is first placed in a separate cluster and then gradually the clusters are merged into larger and larger clusters until all entities are in a single cluster; ii) *optimization algorithms*, where an initial partitioning of the whole system is considered and with iterative entity movements between clusters the clusters are improved to an optimal partition; and iii) *graph-theoretic algorithms*, where an entity relationship graph of the system is considered and the algorithm searches to find subgraphs with special properties such as maximal connected subgraphs or minimal spanning trees [10].

The general form of an iterative partitioning algorithm [34, 10] used in this paper is as follows:

```
Algorithm: iterative partitioning (system files) =
    find an initial partition of $K$ clusters for the system files.
    repeat
        determine the seed point of each cluster.
        move each entity to the cluster with the most similar seed point.
    until no entities were relocated in this iteration.
```

A *seed-point* is an entity in a cluster whose score is an average score of the entities in that cluster. Our approach to an iterative partitioning algorithm is summarized as follows:

- Produce an initial partition of the system files as a number of singleton clusters each containing one seed-point, and the *rest-of-system* (i.e., the remaining files in the system) as a large cluster. The seed-points are distinguished and dissimilar files which are highly associated by other files.
- Perform an optimization operation which iteratively relocates the files (except the seed-points) among different clusters in order to improve the partition according to the group-average-similarity value of the clusters.

The details of the algorithms are discussed below.

8.1. Initial partitioning

The algorithm *initial-partition* (G^C, n) in Figure 7 generates an initial partition from the set of system files N^C using a scoring method to find the seed-points. In order to find a seed-point all the files are tested. The ideal case is to find a group of seed-points whose associated files are completely separated from each other. The first seed-point is the file with the highest total association value of the links attached to it. When a seed-point is selected all its corresponding association links are marked as visited to keep other seed-points apart. The score of the subsequent seed-points decrease if their connected association links have already been visited. The process of finding seed-points stops after finding n seed-points. At this time, each seed-point becomes a singleton cluster and all the rest of files become one cluster called *rest-of-system*. The utility function *get-compAssoc-value*(e_k) returns the annotated *compAssoc* of the link e_k.

8.2. Iterative partitioning

The algorithm *iterative-partitioning*(G^C, n, δ_{sim}) in Figure 8 requires a list of clusters in P to start with. Therefore, either it invokes the algorithm *initial-partition* and receives a list of singleton clusters and the rest-of-system in P, or receives an already computed partition P' whose clusters have been merged, split, or changed. In each iteration, the algorithm computes the average-similarity-value of every file (except the seed-point files) in the clusters to every

Algorithm initial-partition (G^C, n) =

input:
G^C: component graph consisting of all system files N^C and association links R^C.
n: number of singleton clusters.
output:
P: initial partition of the system files N^C into $n+1$ disjoint clusters of files.
local variables:
$\mathcal{L}$: remaining set of system files to be used for partitioning.
L_c, L_x: a candidate file to be tested as seed-point, and a file in N^c.
E: set of all association links to/from a candidate file L_c.
V: set of all already visited association links.
L_{sp}: selected seed-point file.
$score, score_{sp}$: score of a candidate file L_c, and score of the selected seed-point sp.

```
 P := {}    V := {}    L_sp := nil    ℒ := N^C
repeat
    score_sp := 0.0
    for L_c ∈ ℒ do
        score := 0.0
        E := {e_k | e_k ∈ R^C ∧ ∃L_x ∈ N^C • e_k = (L_c, L_x) ∨ e_k = (L_x, L_c)}
        for e_k ∈ E do
            a := get-compAssoc-value(e_k)        % e_k is annotated with a
            if e_k ∉ V then
                score := score + a
             else
                score := score + a/2    % reduce score if e_k is already visited

        if score > score_sp then
            score_sp := score
            L_sp := L_c

    P := P ∪ {{L_sp}}                % {L_sp} is a singleton cluster
    ℒ := ℒ − {L_sp}
    V := V ∪ {e_k | e_k ∈ R^C ∧ ∃L_x ∈ N^C • e_k = (L_sp, L_x) ∨ e_k = (L_x, L_sp)}
    n := n − 1
 until n > 0 do

 P := P ∪ {ℒ}                 % ℒ is now the rest-of-system cluster
 return P
```

Figure 7. Algorithm initial partitioning generates the first partition of clusters.

```
**Algorithm** iterative-partitioning ( $G^C$, $n$, $\delta_{sim}$) =
    **input:**
    $G^C$: component graph consisting of all system files $N^C$ and association links $R^C$.
    $n$: number of clusters excluding the rest-of-system.
    $\delta_{sim}$: min difference of the average-closeness of a file to self-cluster and another cluster.
          This threshold allows to move a file from the self-cluster to another cluster.
    **output:**
    $P$: a list of disjoint clusters of files, as a partition of the system files $N^C$.
    **local variables:**
    $L_c$: a candidate file in the current cluster to be tested for relocation.
    $C_{src}, C_{cur}$ : source cluster whose files are tested against the current cluster $C_{cur}$.
    $sim$, $sim_{src}$, $sim_{max}$: group-average-similarity of a file $L_c$: to current cluster $C_{cur}$;
          to source cluster $C_{src}$; and to a cluster that maximizes the similarity value.
    $aFileMoved$: a flag which is set if in an iteration a single file is moved between clusters.
    **global variables:**
    $P'$: already computed partition $P$ in which one or more clusters merged/split/changed.

        $P :=$ *initial-partition* $(G^C,\ n)$      $\vee$      $P := P'$
        **repeat**
            $aFileMoved :=$ false
            for $i = 1$ *to* $|P|$ do
                $C_{src} := P[i]$
                for $L_c \in (C_{src} - \{\text{seed-points of } C_{src}\})$ *do*        % get candidate file
                    $P[i] := C_{src} - \{L_c\}$
                    $sim_{max} := 0.0$
                    for $j = 1$ *to* $|P|$ do
                        $C_{cur} := P[j]$
                        $sim :=$ *get-average-similarity-value* $(L_c,\ C_{cur},\ G^C)$
                        if $j = i$ then
                            $sim_{src} := sim$
                        if $sim > sim_{max}$ then
                            $sim_{max} := sim$
                            $m := j$                    % $m$ stores the index of destination cluster
                    if $sim_{max} - sim_{src} \geq \delta_{sim}$ then
                        $P[m] := P[m] \cup \{L_c\}$
                        $aFileMoved :=$ true
                    else
                        $P[i] := P[i] \cup \{L_c\}$

        **until** $aFileMoved$
        *return* $P$
```

Figure 8. Algorithm iterative partitioning relocates the files among the clusters according to δ_{sim}.

cluster in the partition P to check if a move to a different cluster is needed or not, and performs accordingly. The test-and-move operation stops when no file is moved between the clusters in an iteration and all files remain in their own clusters, where the computed partition is returned.

The user may investigate the quality of the resulting partition according to the criteria such as *modularity quality* metric or precision/recall against the system documentation. The following operations can be performed if the partition is not satisfactory: i) merge two clusters that are very close; ii) split a large cluster into two clusters with different seed-points; iii) fix some files in particular clusters, so that they will not be moved around. After each of the above operations, the algorithm must be run to rearrange the files into clusters that is optimal with respect to the similarity threshold δ_{sim}. The function *get-average-similarity-value* computes the similarity value of a candidate file to a cluster of files by averaging the *compAssoc* value of the file to every file in the cluster. This value represents the similarity of the candidate file to the average file in that cluster and is equivalent to defining a new seed-point in the general partitioning algorithm discussed earlier.

8.3. Modularity quality evaluation

We use two modularity quality metrics to assess the result of the proposed partitioning technique in section 8.2.

The first metric is defined in equation (8) and measures the modularity quality in terms of intra-/inter-connectivity among the entities in a collection of clusters that form a system partition as discussed in [8]. We refer to this metric as connectivity modularity-quality and denote it as MQ_{con}:

$$MQ_{con} = \frac{1}{k}\sum_{i=1}^{k}\frac{e_i}{N_i^2} - \frac{1}{\frac{k^2-k}{2}}\sum_{i,j=1}^{k}\frac{e_{i,j}}{2N_iN_j} \tag{8}$$

where
k is the number of clusters;
e_i is the number of relations among the functions, datatypes, and variables in a cluster C_i;
$e_{i,j}$ is the number of relations among the functions, datatypes, and variables between two clusters C_i, C_j; and
N_i (N_j) is the number of simple entities in the cluster C_i (C_j).

In equation (8) the first term evaluates the average intra-connectivity among entities in a cluster C_i and the second term evaluates the average inter-connectivity among entities in every two clusters C_i and C_j.

The second metric is defined in equation (9) and measures the association-based modularity quality of a system of files or its partition into clusters and is discussed in [35]. This metric measures the average of difference between "*self-association*" and "*association on/by other clusters*" for a cluster in the partition. The association-based modularity metric is denoted as MQ_{assoc}:

$$MQ_{assoc} = \frac{\sum_{i=1}^{k} \left[compAssoc(C_i, C_i) - \left(\frac{A_i}{\sum_{m=1}^{n_{i,j}} |C_{j_m}|} + \frac{A_i}{|C_i|} \right) \right]}{k} \tag{9}$$

$$\text{such that} \qquad A_i = \sum_{m=1}^{n_{i,j}} compAssoc(C_i, C_{j_m}) * |C_{j_m}|$$

where, k is the number of clusters; A_i is the group-average-similarity between the cluster C_i and other linked clusters C_{j_m} to C_i by merging all linked clusters C_{j_m} into one big cluster; and $n_{i,j}$ is the number of linked clusters C_{j_m} to C_i. In equation (9), the first term in the parentheses computes the average association value of C_i on its linked clusters, and the second term computes the average association value of the linked clusters on C_i.

9. Case studies

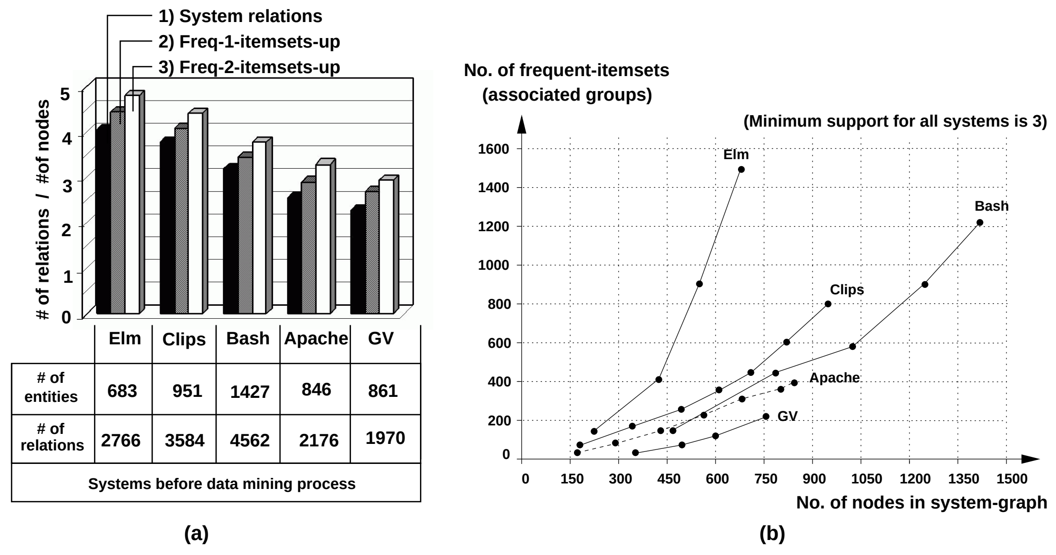
We have implemented an interactive reverse engineering tool (Alborz [15]) to recover the architecture of a software system as cohesive components (i.e., subsystems or modules). The Alborz tool has been built using the Refine re-engineering toolkit [36] and uses the Refine's built-in parsers to parse the software systems.

The Alborz tool supports two clustering techniques, based on either a user-assisted partitioning (discussed in this paper) or a supervised clustering [2]. The latter technique is hierarchical, that is a system is first decomposed into subsystems of files and then each subsystem can be decomposed into modules of functions, datatypes, and variables.

In both techniques, the tool provides metrics to assess the modularity quality of the software system and its decomposition into subsystems or modules. The input to the tool is an information base of entities and relationships of the software system which are extracted from either: i) AST of the software system generated by the Refine's built-in parser; or ii) RSF file generated by the Rigi parser [14]. The tool provides the result of the clustering using: i) HTML pages for the recovered clusters, tool generated metrics, and source code viewing; and ii) graphs of boxes and arrows to be visualized by the Rigi tool, where the boxes are the clusters and the arrows are either "resource interaction" (i.e., import/export of simple entities) or "association links" between clusters.

The experimentations in this section are divided into three parts: first, the application of the Apriori algorithm on the system-graph $G = (N, R)$ and the characteristics of the extracted associated groups are discussed; second, the user/tool collaborative system partitioning provided by the Alborz tool is presented using two software systems; and finally, the evaluation of the proposed partition technique on the basis of two modularity quality measures are discussed.

The experimentations are performed on six middle-size industrial systems, namely: i) *Xfig* drawing editor, ii) *Clips* expert system builder, iii) *Bash* Unix shell; iv) *Apache* http server; v) *Elm* Unix mail system; and vi) *Ghostview* postscript/pdf file viewer and navigator. The experimentations are run on a Sun ultra 10 with 440MHZ CPU, 256M memory, and 512M swap disk space.



	Elm	Clips	Bash	Apache	GV
# of entities	683	951	1427	846	861
# of relations	2766	3584	4562	2176	1970
Systems before data mining process					

Figure 9. (a) The relation-to-node ratio: 1) in the studied software systems before data mining; 2) in the generated frequent 1-itemsets and higher; and 3) in the frequent 2-itemsets and higher. The ratio increases in each subsequent frequent itemsets. (b) The number of generated associated groups versus the number of nodes in the studied software systems. Systems with higher relation-to-node ratio in part (a) generate more associated groups with a given number of nodes.

9.1. Maximal association extraction

In this group of experimentations the *off-line analysis* presented in section 3 is discussed. The groups of entities with maximal association constitute the crucial data for the partitioning process. These data are generated and stored once and are used several times.

In the off-line analysis, the main effort is focused on generating associated groups that include all the relations *use-F, use-T, use-V* having the lowest possible minimum-support value, i.e., 2. Unfortunately, the number of intermediate associated groups are very sensitive to the chosen minimum-support value and for small values the number of groups increases very rapidly hence they require a large swap disk and execution time.

Figure 9(a) illustrates the ratio between the number of relations of types *use-F, use-T, use-V* to the number of nodes of types *function, datatype, variable* in each software system. This ratio is an indication of the overall data/control flow complexity of a system. In the group of bars labeled "system relations" the highest ratio belongs to Elm system with the $\frac{\#relations}{\#nodes} = 4$ and the lowest belongs to Ghostview with ratio 2. The application of Apriori algorithm generates the associated groups of entities from frequent 1-itemsets to frequent k-itemsets where k is the maximum size of the extracted itemsets. As it is seen in Figure 9(a):
ratio for frequent 2-itemsets > ratio for frequent 1-itemsets > ratio for original system.
However, the number of associated groups decrease for a higher itemset size. Since the

associated groups in frequent 1-itemsets have only one shared entity in common, it makes sense to consider frequent 1-itemsets as noise and delete them, hence consider the associated groups in frequent 2-itemsets and up (i.e., two, three, ... entities in common). This causes to compute component association between system files based on only large associated groups of entities, hence producing better partitioning results.

Figure 9(b) illustrates a comparison of the studied systems in terms of the generated groups with maximal association versus number of entities in the systems, all having the relations *use-F, use-T, use-V* and minimum-support 3.

The following observations can be made from the curve of each system in Figure 9(b): i) the rate of generating associated groups is increasing with respect to the size of the entities, where this increase is caused by forming new associated groups whose entities are partly in the newly added entities and partly in the previous entities; ii) systems with higher relation-to-node ratio in Figure 9(a) generate more associated groups with a given number of nodes; and iii) the number of the generated groups are kept within a tractable size by this increase.

Time and space statistics

In Figure 10, the statistics pertinent to the computation time and disk space requirements for the generated associated groups for six systems are presented.

The *minimum-support* number is a control mechanism to reduce the computation time of the Apriori algorithm in generating the frequent itemsets. For the Xfig system, even though the minimum-support threshold is increased to 7 still the maximum size of the generated itemsets is 16 that means large associated groups have been extracted. The combination of the maximum-itemset size and the number of extracted associated group, (i.e., 3167 for Xfig), is a criterion for the user to assess the quality of the generated associated groups. Ideally, we would like to generate the frequent itemsets with minimum-support 2 to take into account all the associated groups. However, for a system with a large number of highly related associated groups, this may cause the number of intermediate frequent itemset to explode. In such cases, still it is possible to obtain enough relations among the system entities by multiple execution of the Apriori algorithm with different minimum-support values, as in case of the Bash system in Figure 10. For the Bash system, the minimum-support 3 produces 1225 associated groups with maximum itemset size 11. However to increase the number of associated groups of entities, the algorithm is executed again with minimum-support 2, but the execution is stopped after generating frequent 4-itemsets, that is before the number of associated groups explodes. In this case, the resulting associated groups produce entity association measure among those entities that did not exist in the previous run of the algorithm with minimum-support 3. Since we stopped the execution, the recovered association values are probably lower than the actually values. In the case of Clips system, the minimum-support is 3 which produces 810 frequent itemsets with different itemset sizes and max-itemset size of 16. This combination is promising for a satisfactory analysis.

9.2. User/tool collaborative system partitioning

In this section the *on-line analysis* in section 3 is discussed using the Xfig system as a case study. Xfig is an interactive drawing editor which runs under the X Windows System and

System	Source: KLOC	No. of Nodes	No. of Relations	Min support (Max itemset)	# Freq−itemsets (stored size)	Time: Freq−item Hour: Min	Graph: # nodes # edges: (S, M, L, W)
Xfig	74	3055	16387	7 (16)	3167 (353KB)	49 : 18	98 (49, 48, 141, 5810)
Clips	40	951	3584	3 (16)	810 (61KB)	8 : 53	44 (22, 24, 64, 944)
Apache	38	846	2176	2 (16)	678 (34KB)	20 : 32	42 (22, 19, 45, 651)
Bash	44	1427	4562	3 (11) 2 (max 4)	1225 (71KB) 18926 (800KB)	0: 03 0 : 46	47 (25, 20, 79, 542)
Elm	35	683	2766	3 (13)	1525 (120KB)	1 : 42	62 (32, 30, 80, 2094)
Ghost view	39	861	1970	3 (15)	411 (21KB)	1 : 04	47 (24, 24, 68, 277)

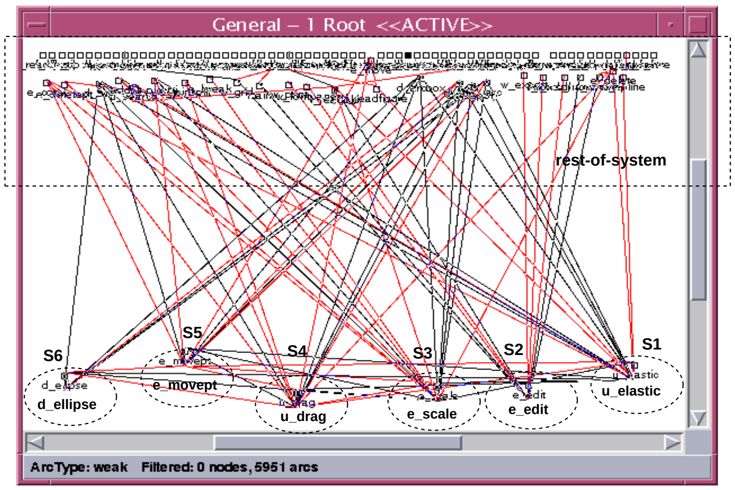
Figure 10. The time and space statistics for generating groups of entities with maximal association. The presented data include: 1) size of the systems in Kilo Lines Of Code (KLOC); 2&3) number of nodes (functions, datatypes, variables) and relations in the system; 4) number of minimum-support and maximum size of itemsets in the generated associated groups; 5) number of generated frequent itemsets (associated groups) and the required disk space to store them; 6) generation time in hours and minutes; and 7) numbers of component graph nodes (files) and edges for (Strong, Medium, Loose, Weak) association strength ranges.

consists of 74 KLOC, 98 source files, 75 include files, 1662 functions, 1356 global variables, and 37 aggregate types.

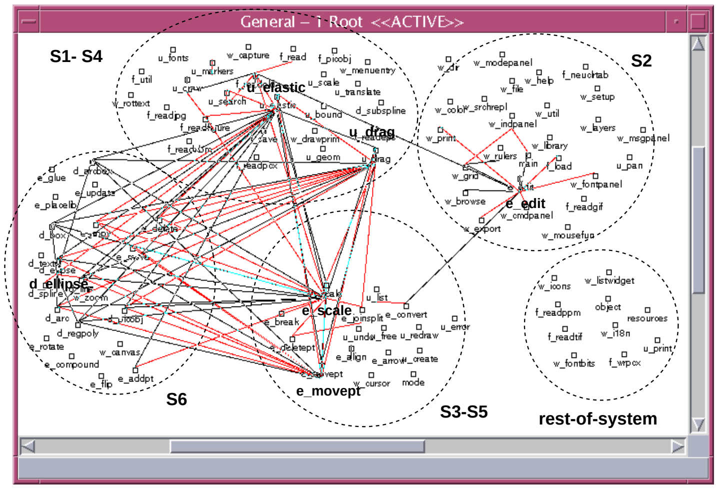
Figure 11 illustrates the partitioning of the quantized component graph G^C using the graph visualization tool Rigi [14]. In these graphs a box is either a system file or a subsystem of files (Figure 11(c)) and a line is a quantized association link to represent the association strengths among the files or subsystems. A line from the bottom of a box L_i to the top of another box L_j represents $compAssoc(L_i, L_j)$. A box L_i with a crossing line from bottom to top represents $compAssoc(L_i, L_i)$.

Figure 11(a) illustrates the initial partition P of the Xfig system by applying the initial partition algorithm (discussed in section 8.1) on the system files N^C. The edges between the Xfig files consist of strong and medium association links. The initial partition includes six singleton clusters, each containing a seed-point, and the *rest-of-system* as a large cluster. The order of the selected seed-points are from S1 to S6. Each seed-point has many links to the files in the rest-of-system which qualifies it to be a seed-point. However, in this case study the number of links between the seed-points is high, indicating high interaction among the resulting subsystems.

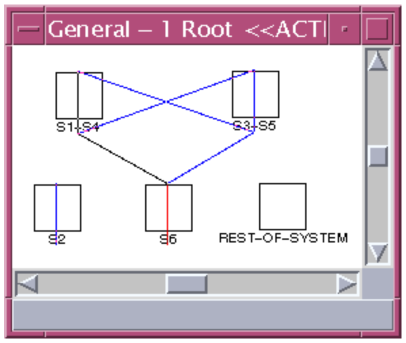
The application of the iterative partitioning algorithm on the initial partition P, discussed in section 8.2, is shown in Figure 11(b). In this partition, the singleton clusters have been populated by moving the similar files from rest-of-system into them. Also, two pairs of clusters have been merged into two cluster S1-S4 and S3-S5. The reason is that both S1 and S4 collect the files from *utility* and *file manipulation* subsystems of Xfig (discussed later), that suggest to merge them into one cluster. The similar reason holds for merging S3 and S5. As a result,



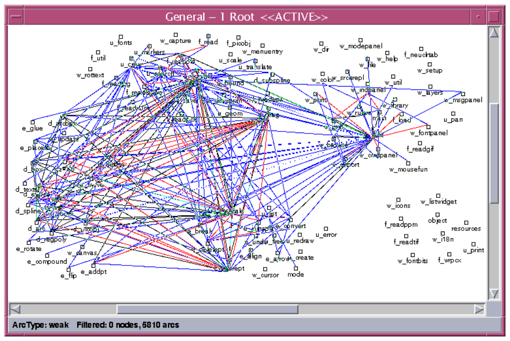
(a) Initial partition of Xfig system: six singleton clusters and rest-of-system are shown using "strong" and "medium" association links.



(b) Final partition of Xfig system: the initial clusters S1, S4 and S3, S5 are merged to generate two clusters S1-S4 and S3-S5.

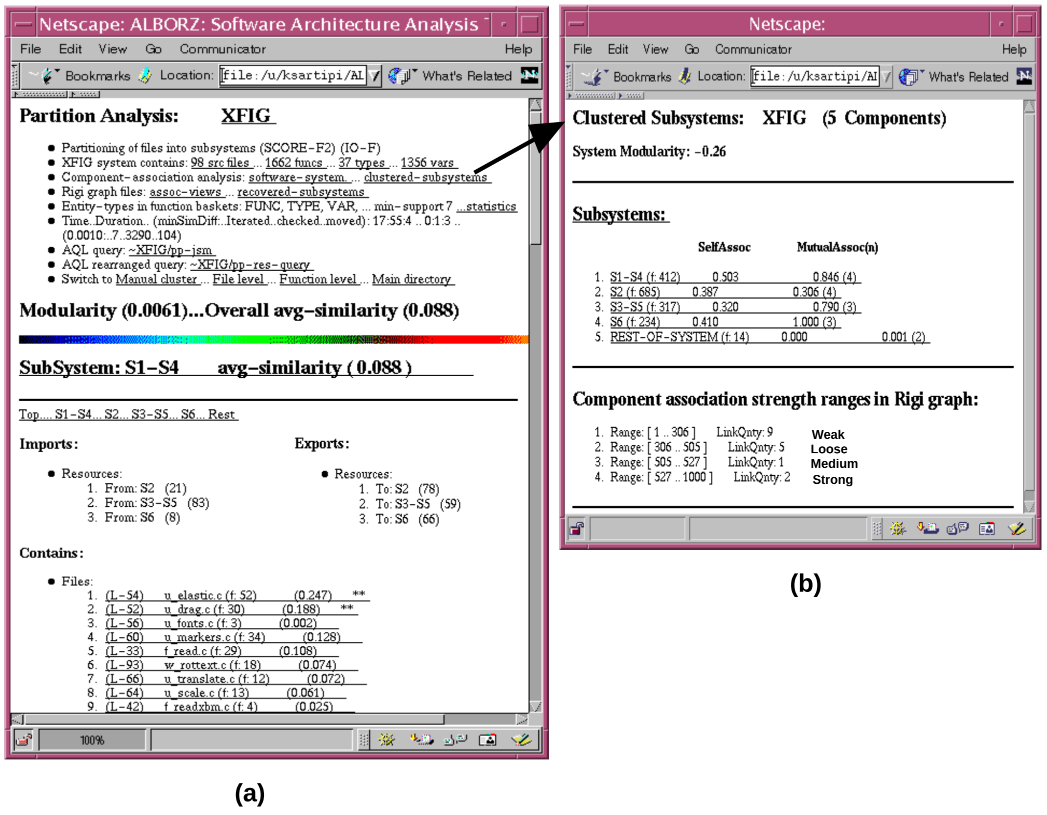


(c) Association links among the resulting subsystems in part (b).


General – 1 Root <<ACTIVE>>
ArcType: weak Filtered: 0 nodes, 5810 arcs

(d) Graph of Xfig system with "strong", "medium", and "loose" associtaion links.

Figure 11. System partitioning of the Xfig system using the quantized component graph G^C.


Netscape: ALBORZ: Software Architecture Analysis
File Edit View Go Communicator Help
Bookmarks Location: file:/u/ksartipi/AL What's Related
100%
(a)
Netscape:
File Edit View Go Communicator Help
Bookmarks Location: file:/u/ksartipi/AI What's Related
(b)

Partition Analysis: XFIG

- Partitioning of files into subsystems (SCORE-F2) (IO-F)
- XFIG system contains: 98 src files ... 1662 funcs ... 37 types ... 1356 vars
- Component-association analysis: software-system ... clustered-subsystems
- Rigi graph files: assoc-views ... recovered-subsystems
- Entity-types in function baskets: FUNC, TYPE, VAR, ... min-support 7 ...statistics
- Time..Duration.. (minSimDiff:..Iterated..checked..moved): 17:55:4 .. 0:1:3 .. (0.0010:..7..3290..104)
- AQL query: ~XFIG/pp-jsm
- AQL rearranged query: ~XFIG/pp-res-query
- Switch to Manual cluster ... File level ... Function level ... Main directory

Modularity (0.0061)...Overall avg-similarity (0.088)

SubSystem: S1-S4 avg-similarity (0.088)

Top.... S1-S4... S2... S3-S5... S6... Rest

Imports:

- Resources:
  1. From: S2 (21)
  2. From: S3-S5 (83)
  3. From: S6 (8)

Exports:

- Resources:
  1. To: S2 (78)
  2. To: S3-S5 (59)
  3. To: S6 (66)

Contains:

- Files:
  1. (L-54) u_elastic.c (f: 52) (0.247) **
  2. (L-52) u_drag.c (f: 30) (0.188) **
  3. (L-56) u_fonts.c (f: 3) (0.002)
  4. (L-60) u_markers.c (f: 34) (0.128)
  5. (L-33) f_read.c (f: 29) (0.108)
  6. (L-93) w_rottext.c (f: 18) (0.074)
  7. (L-66) u_translate.c (f: 12) (0.072)
  8. (L-64) u_scale.c (f: 13) (0.061)
  9. (L-42) f_readxbm.c (f: 4) (0.025)

Clustered Subsystems: XFIG (5 Components)

System Modularity: -0.26

Subsystems:

	SelfAssoc	**MutualAssoc(n)**
1. S1-S4 (f: 412)	0.503	0.846 (4)
2. S2 (f: 685)	0.387	0.306 (4)
3. S3-S5 (f: 317)	0.320	0.790 (3)
4. S6 (f: 234)	0.410	1.000 (3)
5. REST-OF-SYSTEM (f: 14)	0.000	0.001 (2)

Component association strength ranges in Rigi graph:

1. Range: [1 .. 306]	LinkQnty: 9	**Weak**
2. Range: [306 .. 505]	LinkQnty: 5	**Loose**
3. Range: [505 .. 527]	LinkQnty: 1	**Medium**
4. Range: [527 .. 1000]	LinkQnty: 2	**Strong**

Figure 12. The analysis of the clustered subsystems of Xfig system. (a) Resulting system partitioning of Xfig. (b) Subsystem association analysis and its link from the main page.

the Xfig files have been clustered into four subsystems and the rest-of-system cluster based on the association strengths between files.

Figure 11(c) demonstrates the component graph for the partitioned system with strong, medium, and loose association links between the resulting subsystems. This graph is a simple representation of the group of association links in Figure 11(b), where, the subsystems S1-S4 and S3-S5 have strong association with subsystem S6, and the subsystem S2 and *rest-of-system* are isolated. Figure 11(d) illustrates adding loose association links to Figure 11(b), where still the weak association links are not added. It is easily seen how filtering the low association strength links from the component graph can assist the user to investigate the system under analysis by viewing the locus of high interaction among the system files.

HTML pages

The result of partitioning algorithm on the system files is presented by the HTML page *partition analysis* in Figure 12(a). The top part of this page provides overall information about the software system with links to different pages of information, statistics about the data

Clustered subsystems	No. of files	Xfig subsystems	No. of files	Precision	Recall
S1–S4	26	utility & file manipulation	32	81%	56% u– 78% f–
S2	25	X–windowing	28	80%	72% w–
S3–S5	15	editing & utility	37	94%	42% e– 33% u–
S6	21	editing & drawing	29	90%	53% e– 90% d–
rest–of–sys	10	————	——	——	——

Figure 13. The evaluation of the Xfig system partitioning using the Precision and Recall metrics.

mining results, statistics about the partitioning algorithm execution, and evaluation metrics as was discussed in section 8.3. The bottom part of the page corresponds to the five clustered subsystems S1-S4, S2, S3-S5, S6, and *rest-of-system*. For each subsystem, the *imports/exports* parts represent the interactions of the subsystems through simple entities in the form of group of resources (or individual resources with links to source code). For example, subsystem S1-S4 imports 21 functions from $S2$. Each file of subsystem S1-S4 is shown in a separate line with the closeness value to other files in that subsystem, e.g., 0.247 for file *u_elastic.c* in line 1.

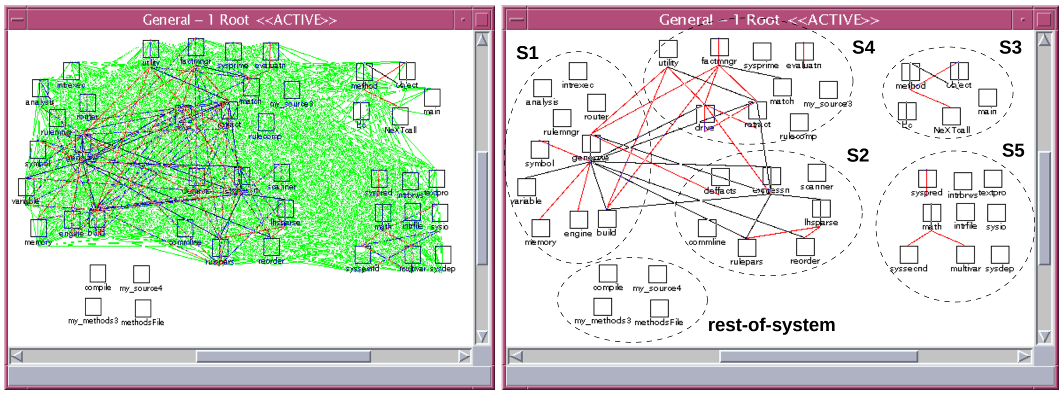
The association statistics and overall achieved modularity measure of the clustered subsystems are provided via the HTML page *clustered subsystems* in Figure 12(b). In this page, the association value of each subsystem on itself (*self-association*) and the total association value of each subsystem on other subsystems that are linked to it (*mutual association*(n)) are shown, where n is the number of linked subsystems and *mutual association* means that the association value is the same in both directions. For example, line 1 that is shown below is interpreted as: subsystem S1-S4 with 412 functions has medium *SelfAssoc* value, and has high *MutualAssoc* value on other subsystems.

	SelfAssoc	MutualAssoc(n)
1. S1-S4 (f:412)	0.503	0.846(4)

Xfig partitioning evaluation

According to personal communication with the maintainer of the Xfig system [37], Xfig lacks any documentation on the structure or implementation, and only the user manual exist. However, a consistent naming convention is used throughout the system files which can be referred as the structure of the system. The system naming conventions are as follows: *d_** files relate to drawing shapes; *e_** files relate to editing shapes; *f_** files have file-related procedures; *u_** files are utilities for creating or editing shapes ; and *w_** files have X11 window calls in them to do all of the window-related functions.

Figure 13 presents the evaluation of the Xfig system partitioning using the information retrieval metrics *Precision* and *Recall*. The result of partitioning conforms with the above task



Clustered subsystems	No. of files	Clips subsystems (documented)	No. of files	Precision	Recall
S1	10	Parsing & System support	11	70%	64%
S2	7	Expression evaluation	4	43%	75%
S3	5	Object	4	80%	100%
S4	9	Inference engine	6	55%	84%
S5	9	Rule manipulation & user interface	9	88%	88%
rest-of-sys	4	———	———	———	———

(c)

Figure 14. System partitioning and evaluation of the Clips system. (a) Component graph with all association links. (b) System partition into six clusters. (c) Partitioning evaluation using *Precision* and *Recall* metrics.

description of the Xfig files. The result shows that two subsystems *editing* shapes and *utility* functions are partly clustered into two different subsystems each shown with the corresponding Recall value. The reason is that the functionality of the drawing shape files and editing shape files are closely related (subsystem S6) and the utility files provide services for drawing and editing files (subsystem S3-S5). The obtained Precision and Recall values indicate that partitioning process has recovered the Xfig subsystems with high accuracy.

9.3. System partitioning of the Clips system

The Clips system provides an environment for construction of the rule based expert systems and is supported by an architectural manual [38] which is our reference in this experimentation. Clips consists of 40 KLOC, 44 source files, 736 functions, 161 global variables, and 54 aggregate types.

The application of the iterative partitioning algorithm on the Clips system is shown in Figures 14(a) and (b). The correspondences of the partitioning result with the documentation

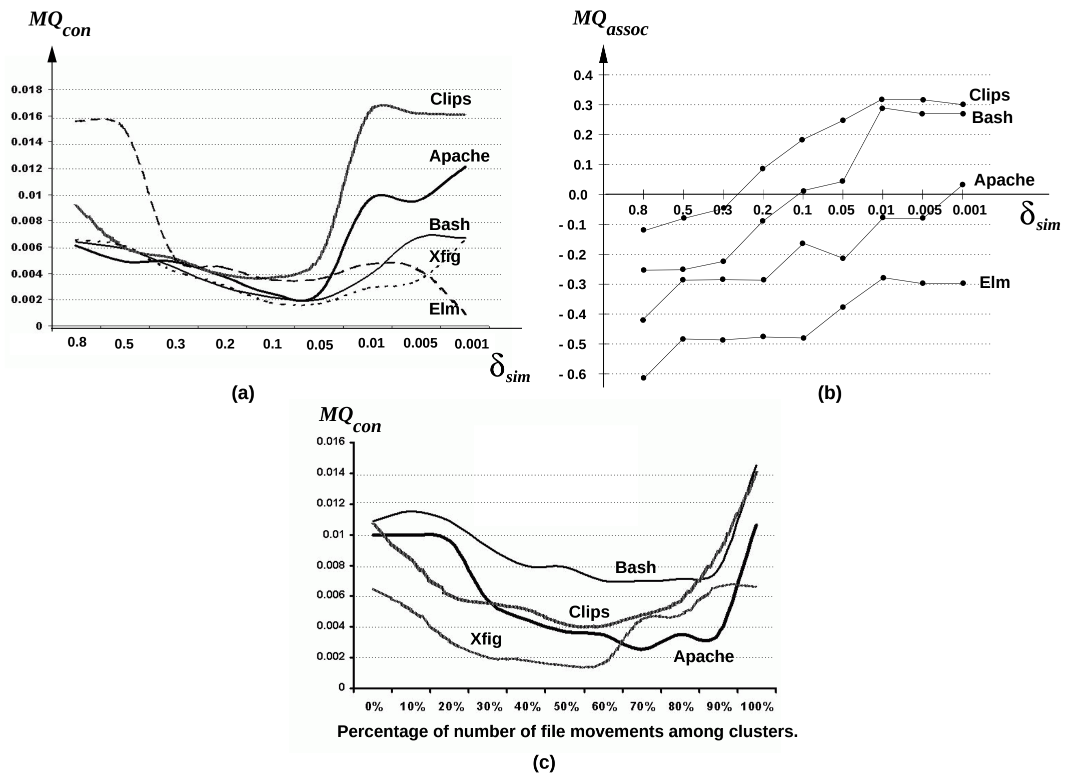


Figure 15. The modularity-quality measure of five partitioned systems based on (a) inter-/intra-connectivity among the clusters, and (b) association among the clusters. (c) The connectivity-based modularity measure versus the number of file movements among the clusters.

of the Clips system in terms of Precision and Recall metrics are presented in Figure 14(c), which is considered as a very promising result.

9.4. Modularity quality evaluation

Figures 15(a) and (b) illustrate the performance of the proposed partitioning algorithm in increasing the modularity quality of five partitioned systems based on two modularity quality metrics defined in section 8.3. Each system is partitioned into four clusters and the modularity values are measured versus the similarity threshold δ_{sim}, where δ_{sim} is the similarity-difference of a file to two clusters that determines whether a file moves between two clusters or not.

Three regions are considered in Figure 15(a) as follows: I) For large values of δ_{sim} (i.e., 0.8 to 0.3) only a few files with high closeness values to the clusters are moved from the rest-of-system, and the majority of the files remain in the rest-of-system. When highly close files exist in each cluster the amount of intra-cluster interaction is high compared to inter-cluster interaction, hence, the value of MQ_{con} is high for large values of δ_{sim}. II) For medium values of δ_{sim} (i.e, 0.3 to 0.05) the inter-cluster interaction increases since more files are moved to clusters, however the quality of the clusters may not be sufficiently improved. This caused a drop in the value of MQ_{con}. III) For small values of δ_{sim} (i.e, 0.05 to 0.001) the quality of the clusters improve to accommodate groups of highly close files into the clusters, hence the value of MQ_{con} increases again.

Figure 15(b) illustrates the same experimentation discussed above with respect to the association-based modularity metric MQ_{assoc}. The value of MQ_{assoc} monotically increases from the initial-partition of the system (range I) to its final state (range III). This indicates that MQ_{assoc} considers both size and quality of the clusters, hence, evaluates a "low" modularity value for the initial partitioning of the system, as opposed to MQ_{con}.

Figure 15(c) presents the improvement of the modularity value MQ_{con} during the execution of the iterative partitioning algorithm. The modularity of the partitioning changes similar to the experimentation in Figure 15(a) discussed above.

Therefore, according to both modularity metrics MQ_{con} and MQ_{assoc} the proposed partitioning technique enhances the modularity value of the partitioned system.

9.5. Discussion

The quality of the resulting partition and the execution time of the iterative partitioning technique discussed in this paper is controlled by the similarity-threshold parameter δ_{sim}. As tested, in most cases the algorithm finds an optimal result (i.e., $\delta_{sim} = 0$) with a given initial partition after a small number of iterations. However, the tool allows the user to stop the algorithm and check the partition if some files are relocated repeatedly among the clusters. For this paper, we examined the proposed algorithm with several middle-size systems (-100 KLOC), however the empirical results show that the algorithm will also terminate in a reasonable time for larger systems, since in most cases after a few iterations the algorithm produces result. As discussed in section 9.1 for data mining statistics, producing associated groups with the lowest possible minimum-support value 2 is not always feasible for the large systems. This affects the quality of the partitioning because increasing the minimum-support deletes some association relations among the system entities. However, the tool provides means for merging the results of data-mining with different minimum-support values, or merging the results of data mining for different relations *use-F, use-T*, and *use-V*. These techniques recover all the association relations that had been missing in the first execution of data mining, but the obtained association values for the missing relations may be less than the real values that actually exist. The main input for the proposed approach is an entity-relation database of the software system according to the abstract domain model discussed in section 4. Therefore, the approach is not programming language dependent. For the experimentation purposes, currently we use Refine's built-in C parser to parse systems written in C, however, the tool can analyze the systems whose entities and relationships are presented as RSF format. The approach can also be extended to analyze object-oriented systems by defining a similar entity relation domain model for object-oriented systems.

10. Conclusion

Software systems evolve over time and their original design is constantly modified to reflect the result of a series of corrective, perfective, or enhancing maintenance activities. In this paper, we argue that recovering the design of such heavily modified systems requires a user-assisted iterative and incremental reverse engineering process that can augment the automated recovery techniques. Specifically, we presented a user-assisted architectural design extraction methodology which we believe it is suitable for the recovery of cohesive subsystems based on an iterative partitioning clustering in order to obtain higher quality design for the system. In such a design recovery environment, the tool supplies pre-processed system information, using data mining and association metrics, to enable the user to obtain insight into the design of the system. Entity and component association metrics measure the maximal association among the system entities or components as the means to cluster the components into subsystems. Finally, a quantization heuristic converts the broad range of the association values among the components into four ranges, which facilitate the system graph visualization and clustering process. Experimentation with six middle size systems provided an evaluation of the proposed approach with respect to the accuracy of the proposed approach.

The next steps for this research include the investigation of constraint-based and pattern-based clustering where the user imposes certain constrained criteria in the form of a query or a pattern that the clustering result should comply with. Moreover, we would like to investigate requirement-driven clustering techniques where the clustering process is fine-tuned for obtaining a system partition that complies with specific non-functional requirements (e.g., modularity, adaptability).

REFERENCES

1. Wallmuller E. *Software quality assurance: A practical approach.* Prentice Hall: New York NY, 1994; 1–3.
2. Sartipi K, Kontogiannis K. Component clustering based on maximal association. *Proceedings of the Working Conference on Reverse Engineering*, WCRE'01. IEEE Computer Society: Los Alamitos CA, 2001; 103–114.
3. Koschke R. An incremental semi-automatic method for component recovery. *Proceedings of the Sixth Working Conference on Reverse Engineering*, WCRE'99. IEEE Computer Society: Los Alamitos CA, 1999; 256–267.
4. Anquetil N, Lethbridge TC. Experiments with clustering as a software remodularization. *Proceedings of the Sixth Working Conference on Reverse Engineering*, WCRE'99. IEEE Computer Society: Los Alamitos CA, 1999; 235–255.
5. Canfora G, Czeranski J, Koschke R. Revisiting the delta ic approach to component recovery. *Proceedings of the Working Conference on Reverse Engineering*, WCRE'00. IEEE Computer Society: Los Alamitos CA, 2000; 140–149.
6. Davey J, Burd E. Evaluating the suitability of data clustering for software remodularization. *Proceedings of the Seventh Working Conference on Reverse Engineering*, WCRE'00. IEEE Computer Society: Los Alamitos CA, 2000; 268–276.
7. Van Deursen A, Kuipers T. Identifying objects using cluster and concept analysis. *Proceedings of the International Conference on Software Engineering*, ICSE'99. ACM Press: New York NY, 1999; 246–255.
8. Mancoridis S, *et al.* Using automatic clustering to produce high-level system organizations of source code. *Proceedings of the International Workshop on Program Comprehension*, IWPC'98. IEEE Computer Society: Los Alamitos CA, 1998; 45–53.

9. Tzerpos V, Holt RC. Software botryology: Automatic clustering of software systems. *Proceedings of the International Workshop on Database and Expert Systems Applications.* IEEE Computer Society: Los Alamitos CA, 1998; 25–28.
10. Wiggerts TA. Using clustering algorithms in legacy systems modularization. *Proceedings of the Fourth Working Conference on Reverse Engineering*, WCRE'97. IEEE Computer Society: Los Alamitos CA, 1997; 33–43.
11. Hutchens DH, Basili VR. System structure analysis: Clustering with data bindings. *IEEE Transactions on Software Engineering* 1985; **11**(8):749–757.
12. Kunz T, Black JP. Using automatic process clustering for design recovery and distributed debugging. *IEEE Transactions on Software Engineering* 1995; **21**(6):515–527.
13. Finnigan P, *et al.* The software bookshelf. *IBM Systems Journal* 1997; **36**(4):564–593.
14. Muller HA, *et al.* A reverse-engineering approach to subsystem structure identification. *Journal of Software Maintenance: Research and Practice* 1993; **5**(4):181–204.
15. Sartipi K. Alborz: A query-based tool for software architecture recovery. *Proceedings of the IEEE International Workshop on Program Comprehension*, IWPC'01. IEEE Computer Society: Los Alamitos CA, 2001; 115–116.
16. Siff M, Reps T. Identifying modules via concept analysis. *IEEE Transactions on Software Engineering* 1999; **25**(6):749–768.
17. Lindig C, Snelting G. Assessing modular structure of legacy code based on mathematical concept analysis. *Proceedings of the 19th International Conference on Software Engineering*, ICSE'97. ACM Press: New York NY, 1997; 349–359.
18. Tzerpos V, Holt RC. Acdc: An algorithm for comprehension-driven clustering. *Proceedings of the Seventh Working Conference on Reverse Engineering*, WCRE'00. IEEE Computer Society: Los Alamitos CA, 2000; 258–267.
19. Ferneley EH. Design metrics as an aid to software maintenance: An empirical study. *Journal of Software Maintenance: Research and Practice* 1999; **11**(1):55–72.
20. Lakhotia A. A unified framework for expressing software subsystem classification techniques. *Journal of Systems and Software* 1997; **36**(3):211–231.
21. Eshera MA, Fu KS. A graph distance measure for image analysis. *IEEE Transactions on Systems Man and Cybernetics* SMC 1984; **14**(3):398–408.
22. Stevens WP, Myers GJ, Constantine LL. Structured design. *IBM Systems Journal* 1974; **13**(2):115–139.
23. Yourdon E, Constantine LL. *Structured Design.* Yourdon Press: New York NY, 1976; 76–126.
24. Pressman RS. *Software Engineering, A Practitioner Approach.* McGraw-Hill, third edition, 1992; 334–338.
25. Chapin N. Coupling and strength, a la Harlan D. Mills. *Proceedings of the Science and Engineering for Software Development.* IEEE Computer Society: Los Alamitos CA, 1999; 4–13.
26. Lakhotia A. Rule-based approach to computing module cohesion. *Proceedings of the International Conference on Software Engineering*, ICSE'93. ACM Press: New York NY, 1993; 35–44.
27. Nandigam J, Lakhotia A, Cech CG. Experimental evaluation of agreement among programmers in applying the rules of cohesion. *Journal of Software Maintenance: Research and Practice* 1999; **11**(1):35–53.
28. Bieman JM, Ott LM. Measuring functional cohesion. *IEEE Transactions on Software Engineering* 1994; **20**(8):644–657.
29. Misic VB. Coherence equals cohesion-or does it? *Proceedings of the Seventh Conference on Asia-Pacific Software Engineering.* IEEE Computer Society: Los Alamitos CA, 2000; 465–469.
30. Patel S, Chu W, Baxter R. A measure for composite module cohesion. *Proceedings of the International Conference on Software Engineering*, ICSE'92. ACM Press: New York NY, 1992; 38–48.
31. Agrawal R, Srikant R. Fast algorithm for mining association rules. *Proceedings of the 20th International Conference on Very Large Databases*, VLDB. Morgan Kaufmann: San Fransisco, CA, 1994; 487–499.
32. Ramakrishnan R, Gehrke J. *Database Management Systems.* McGraw-Hill, second edition, 2000; 707–735
33. Tzerpos V, Holt RC. The orphan adoption problem in architecture maintenance. *Proceedings of the Working Conference on Reverse Engineering*, WCRE'97. IEEE Computer Society: Los Alamitos CA, 1997; 76–82.
34. Jain AK. *Algorithms for Clustering Data.* Prentice Hall: Englewood Cliffs, N.J., 1988; 89–92.
35. Sartipi K. A software evaluation model using component association views. *Proceedings of the IEEE International Workshop on Program Comprehension*, IWPC'01. IEEE Computer Society: Los Alamitos CA, 2001; 259–268.
36. *Refine User's Guide, version 3.0.* Reasoning Systems Inc.: Palo Alto CA, 1990; 340 pp.

37. Smith BV. *Xfig architecture.* Personal e-mail correspondence with the maintainer of Xfig, September 2000.
38. AI Group. *CLIPS Architectural Manual Version 4.3.* Lyndon B. Johnson Space Center, report No. jsc-23047, 1989; 136 pp.

AUTHOR'S BIOGRAPHY

Kamran Sartipi obtained a M.Sc. degree at the department of Electrical and Electronic Engineering, University of Tehran, Tehran, Iran, and a PhD degree at the School of Computer Science, University of Waterloo, Waterloo, Canada. Kamran has been involved in research and teaching in both hardware and software engineering. In recent years, his research included different aspects of software architecture such as software architecture views, recovery, and evaluation. He has designed and developed techniques and a toolkit for pattern-based software software architecture recovery and evaluation. This research was funded by IBM Canada Ltd. Laboratory - Center for Advanced Studies (Toronto), the Consortium for Software Engineering Research (CSER), and the Institute for Robotics & Intelligent Systems (IRIS).

AUTHOR'S BIOGRAPHY

Kostas Kontogiannis is an Associate Professor at the Department of Electrical and Computer Engineering, University of Waterloo. Kostas obtained a B.Sc in Applied Mathematics from the University of Patras, Greece, a M.Sc in Computer Science from Katholieke Universiteit Leuven, Belgium, and a Ph.D degree in Computer Science from McGill University, Canada. Kostas is leading the Software Re-engineering group at the Department of Electrical & Computer Engineering. He is actively involved in various projects with IBM Canada, Center for Advanced Studies, the Network of Centers of Excellence, Bell Mobility, and the Consortium for Software Engineering Research. Kostas' research is focusing on the design and implementation of tools and techniques that allow for large legacy software systems to be analyzed and integrated in distributed and Network-Centric environments. Application areas include reverse engineering, architectural recovery, software migration, system integration, and distributed object technologies. He is also a recipient of a Canada Foundation for Innovation Award and, the IBM University Partnership Program Awards for the years 2001 and 2002.