Information and Software Technology 53 (2011) 601–614

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Applying agglomerative hierarchical clustering algorithms to component identification for legacy systems

Jian Feng Cui [a], Heung Seok Chae [b,*]

[a] Department of Computer Science and Technology, Xiamen University of Technology, 600 LiGong Rd., Xiamen 361024, China
[b] Department of Science and Engineering, Pusan National University, 30 Changjeon-dong, Keumjeong-gu, Busan 609-735, South Korea

ARTICLE INFO

Article history:
Received 23 January 2010
Received in revised form 9 January 2011
Accepted 15 January 2011
Available online 23 January 2011

Keywords:
Component identification
Agglomerative hierarchical clustering algorithm
Weighting scheme
Similarity measure
Legacy systems
Software reengineering

ABSTRACT

Context: Component identification, the process of evolving legacy system into finely organized component-based software systems, is a critical part of software reengineering. Currently, many component identification approaches have been developed based on agglomerative hierarchical clustering algorithms. However, there is a lack of thorough investigation on which algorithm is appropriate for component identification.
Objective: This paper focuses on analyzing agglomerative hierarchical clustering algorithms in software reengineering, and then identifying their respective strengths and weaknesses in order to apply them effectively for future practical applications.
Method: A series of experiments were conducted for 18 clustering strategies combined according to various similarity measures, weighting schemes and linkage methods. Eleven subject systems with different application domains and source code sizes were used in the experiments. The component identification results are evaluated by the proposed size, coupling and cohesion criteria.
Results: The experimental results suggested that the employed similarity measures, weighting schemes and linkage methods can have various effects on component identification results with respect to the proposed size, coupling and cohesion criteria, so the hierarchical clustering algorithms produced quite different clustering results.
Conclusions: According to the experimental results, it can be concluded that it is difficult to produce perfectly satisfactory results for a given clustering algorithm. Nevertheless, these algorithms demonstrated varied capabilities to identify components with respect to the proposed size, coupling and cohesion criteria.

1. Introduction

Legacy systems are software systems that cannot be easily dealt with but are vital to our organizations [1]. Legacy systems continue to be used because of the prohibitive cost of replacing or redesigning them, despite their poor competitiveness and compatibility with modern equivalents [2]. The evolution of a legacy system is a complex task, which is influenced by several concerns (e.g. its decomposability, budget, technical and time constraints, etc.) [3]. A key activity in software reengineering consists of gathering the software entities comprising the system into meaningful (highly cohesive) and independent (loosely coupled) components; this is called component identification. The common process of component identification starts by parsing the source code of a legacy system, and then the source code is organized into cohesive subsystems that are loosely interconnected by a particular clustering algorithm [11]. According to [5], a component specifies a formal contract of services that it provides to its clients and those that it requires from other components or services in the system in the terms of its interfaces. In terms of object-oriented software systems, a component consists of a set of member classes and interfaces which specify their services.

Agglomerative hierarchical clustering algorithms have been applied in many existing component identification approaches, because a multi-level architectural view produced by agglomerative hierarchical clustering algorithms facilitates architectural understanding [4,7,8,10]. Understanding the behaviors of employed clustering algorithms is the first important step for meaningful utilization of clustering techniques for component identification [6]. The similarity measure and linkage method are the two most important factors in the agglomerative hierarchical clustering algorithms studied by many researchers. In object-oriented systems, the calculation of connection strength between software entities, which is also known as weighting scheme, are affected by the connection manners between software entities. Weighting

* Corresponding author. Tel.: +82 51 510 3517; fax: +82 51 515 2208.
E-mail address: hschae@pusan.ac.kr (H.S. Chae).

schemes can have significant effects on clustering behaviors, but most of them just focused on binary feature values [7,8,10,35]. In this paper, we classified weighting schemes as binary weighting scheme, absolute weighting scheme and relative weighting scheme. We performed a series of experiments to examine their performances.

Agglomerative hierarchical clustering algorithms have been widely employed in software clustering techniques, however, many researches merely focused on proposing an approach to solving a particular problem [16]. In this paper, we performed a study on evolving object-oriented legacy systems into component-based systems by using a variety of agglomerative hierarchical clustering methods which employed various similarity measures, linkage methods and weighting schemes. The goal of this paper is to examine whether there is a superior method in the clustering process. A series of experiments were conducted with various clustering algorithms for several object-oriented legacy systems. Based on the clustering results, we provided an evaluation of the relative strengths and weaknesses of various agglomerative hierarchical clustering algorithms by using a set of criteria with respect to the size, coupling and cohesion criteria. Our focus is on analyzing agglomerative hierarchical clustering algorithms in software reengineering, and then identifying their respective strengths and weaknesses in order to apply them effectively for future practical applications.

The rest of this paper is organized as follows. Section 2 gives an overview of agglomerative hierarchical clustering algorithms. Section 3 illustrates the similarity measures and weighting schemes used for agglomerative hierarchal clustering. Section 4 presents the experimental settings: the tool CIETool developed for automating the process of component identification and evaluation, the clustering strategies, the criteria used for evaluating the clustering results and the employed subject systems. In Section 5, we present the clustering results and evaluate them with the proposed size, coupling and cohesion criteria. Section 6 discusses related work. Finally, Section 7 summarizes our study and provides suggestions for future work.

2. An overview of agglomerative hierarchical clustering algorithms

In this section, we provide an overview of agglomerative hierarchical clustering algorithms and illustrate how they are utilized for software component identification. The agglomerative hierarchical clustering algorithms (AHCA) start from a set of individual entities that are first grouped into small clusters; these are in turn grouped into larger clusters until reaching a final all inclusive clustering [8]. One advantage of these algorithms is that they are non-supervised. They do not need any extra information such as the number of expected clusters and candidate regions of search space for locating each cluster. AHCA provides a view for software clustering; the earlier iterations present a detailed view of the software architecture and the later ones present a high-level view.

When AHCA is utilized for software component identification, the first problem that needs to be addressed is to determine the types of entities to be clustered. In this study, our aim is to reengineer an object-oriented legacy system into a component-based system. Classes are treated as the entities to be clustered, because they are the fundamental parts comprising an object-oriented software system. Interface classes and abstract classes are ignored, because they generally contribute to system structures construction rather than concrete functions realization. The inner class is also ignored, because it is nested in a regular outer class and it is naturally viewed as a member of latter. In this paper, we call the types of classes under study *normal classes*. For simplicity, all usage of the term *class* refers to the normal classes hereinafter unless otherwise indicated in this paper.

Generally a software system is composed of a set of classes and various relations. Classes can be thought of as the nodes in a graph and their relations as the edges in this graph. Fig. 1 shows an example system represented by an undirected graph.

Classes can be connected by various relations within a software system. In this study, we define two classes cls_i and cls_j to be connected if and only if at least one of the following connections exists between them:

- *attr* is an attribute of cls_i, and cls_j is the type of *attr*;
- *op* is an operation of cls_i, and cls_j is the type of a parameter of *op*, or the return type of *op*;
- *op* is an operation of cls_i, and cls_j is the type of a local variable of *op*;
- *op* is an operation of cls_i, and cls_j is the type of a parameter of an operation invoked by *op*;
- *op* is an operation of cls_i, *attr* is an attribute of cls_j, and *op* references *attr*;
- *op* is an operation of cls_i, op' is an operation of cls_j, and *op* invokes op'.

In addition, the classes can be distinguished by their invocation directions. For example, in the above statements, cls_i is the accessing class, and cls_j is the accessed class. Considering inheritance, in this study, we just consider implemented attributes and operations for classes. That is, a parent class connecting some class does not necessarily mean that its child class also connects that class.

Throughout this paper the following notations are used to describe a software system:

- Assuming that *UC* denotes the set of classes in a software system *S*, then $|UC|$ is the number of classes of *S*.
- Classes can be grouped into different components. Let *C-SET(cmp)* denote the set of classes in the comp*onent cmp. We suppose that* $CSET(cmp_m) \cap CSET(cmp_n) = \emptyset$, *if* $m \neq n$.
- A software system can be partitioned into a set of components. Let *UCMP* denote the set of components that compose a software system. Suppose that the system *S* is composed of *t* components, then $|UCMP| = t$.
- Let $conn(cls_i, cls_j)$ denote the set of connections between classes cls_i and cls_j, where cls_i is the accessing class and cls_j is the accessed class.
- Let $conn(cmp_i, cmp_j)$ denote the set of connections between components cmp_i and cmp_j, where cmp_i is the accessing component and cmp_j is the accessed component, which are determined by the invocation direction of the classes in the corresponding components.

Fig. 2 illustrates clustering of AHCA. AHCA accepts the set of individual classes *UC* of the software system *S* as the input. A given similarity measure is necessary to quantitatively calculate the similarity between each pair of classes. *UCMP* is the set of components. Initially, each element of *UCMP* contains an individual class of the

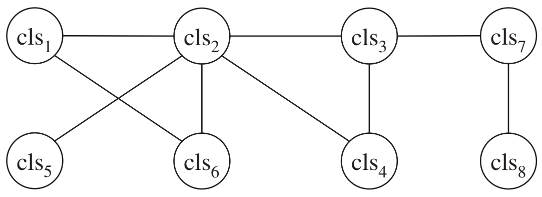

Fig. 1. An example system.