The Journal of Systems and Software 133 (2017) 28–53

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Automatic clustering constraints derivation from object-oriented software using weighted complex network with graph theory analysis

Chun Yong Chong[a], Sai Peck Lee[b,*]

[a] School of Information Technology, Monash University Malaysia, Jalan Lagoon Selatan, 47500 Bandar Sunway, Selangor Darul Ehsan, Malaysia

[b] Department of Software Engineering, Faculty of Computer Science and IT, University of Malaya, 50603 Lembah Pantai, Kuala Lumpur, Malaysia

ARTICLE INFO

Article history:
Received 12 July 2016
Revised 18 May 2017
Accepted 10 August 2017
Available online 12 August 2017

Keywords:
Constrained clustering
Software clustering
Software remodularisation
Graph theory
Complex network

ABSTRACT

Constrained clustering or semi-supervised clustering has received a lot of attention due to its flexibility of incorporating minimal supervision of domain experts or side information to help improve clustering results of classic unsupervised clustering techniques. In the domain of software remodularisation, classic unsupervised software clustering techniques have proven to be useful to aid in recovering a high-level abstraction of the software design of poorly documented or designed software systems. However, there is a lack of work that integrates constrained clustering for the same purpose to help improve the modularity of software systems. Nevertheless, due to time and budget constraints, it is laborious and unrealistic for domain experts who have prior knowledge about the software to review each and every software artifact and provide supervision on an on-demand basis. We aim to fill this research gap by proposing an automated approach to derive clustering constraints from the implicit structure of software system based on graph theory analysis of the analysed software. Evaluations conducted on 40 open-source object-oriented software systems show that the proposed approach can serve as an alternative solution to derive clustering constraints in situations where domain experts are non-existent, thus helping to improve the overall accuracy of clustering results.

1. Introduction

Maintenance of existing software requires plenty of time in analysing and comprehending the available source code and software documentation. Successful accomplishment of software maintenance is highly dependent on how much information can be extracted by software maintainers. However, due to prolonged maintenance and software updates, the architectural design of software system tends to deviate away from the original design, causing further difficulties in software maintenance. Recovering the architecture of software is therefore an important step to aid in software maintenance. In general, software architecture recovery aims to extract a high-level representation of the architectural information from low-level software artifacts, such as source code, to ensure the fulfilment of requirements, identification of reusable software components, and estimation of cost and risks associated to any changes in requirements (Maqbool and Babri, 2007; Riva, 2000).

Software clustering has received a substantial attention in recent years due to its capability to help in recovering a semantic representation of the software design, which directly aid in software architecture recovery (Maqbool and Babri, 2006, 2007). However, software clustering is typically conducted in an unsupervised manner where software maintainers have no influence on the end results because the effectiveness of software clustering depends greatly on the algorithm used. In the case if software maintainers do not agree with the outcome, they will need to repeat the process again using a different set of configuration and clustering algorithm.

Hence, an improvement to classic unsupervised clustering approaches was proposed in the work by Basu et al. (2004), commonly referred as semi-supervised clustering or constrained clustering, where side information is integrated to further improve the accuracy of clustering results. In the domain of software clustering, semi-supervised approaches typically use small samples of software modules with known cluster assignment which enhances the process of model training (clustering process) with software modules for which the cluster information is not available. The side information, which is commonly referred as clustering constraints that reveal the similarity between pairs of clustering entities or user preferences about how those entities should be grouped during clustering, can be originated explicitly from the domain expert or implicitly from the background knowledge of the prob-

* Corresponding author.
E-mail addresses: chong.chunyong@monash.edu (C.Y. Chong), saipeck@um.edu.my (S.P. Lee).

http://dx.doi.org/10.1016/j.jss.2017.08.017
0164-1212/© 2017 Elsevier Inc. All rights reserved.

lem domain. The clustering constraints may impose certain restrictions such as forcing a pair of clustering entities to be always grouped into the same cluster, or separated into disjoint clusters. These constraints are commonly referred as must-link (ML) and cannot-link (CL) constraints respectively. It has been proven in several fields of research that even some minimal supervision can improve the reliability and accuracy of clustering results (Davidson and Ravi, 2009).

To help reveal the structure and behaviour of software systems, domain experts can exert their opinions in the form of ML or CL constraints to alter the clustering process. However, manually supervising and providing clustering constraints is costly and time consuming (Wagstaff, 2007) for large and complex software systems. Identifying relationships between software components of a software that contains thousand or million lines of code would require a significant amount of time and effort to read all of them carefully. Besides that, most of the time, software maintainers are not directly involved in the early stage of software design especially if the maintenance tasks is outsourced to a third party company. The situation is even worse if the software is poorly designed or the software documentation is not up-to-date, which is common for systems developed in an ad-hoc manner. While most of the existing studies often assumed that feedbacks from domain experts are always readily available, the same assumption cannot be applied in software development especially when dealing with poorly designed or poorly documented software systems.

Most of the existing studies require access to domain experts or a small set of clustering constraints (supervised labelled data) as a pre-requisite. Although feedbacks or supervision by domain experts are useful, one important and non-trivial research question remains open, i.e. how to retrieve clustering constraints if experts are not confident with the constraints given or such expertise is not available. While various studies have shown that a small number of constraints can greatly improve the result of clustering, most of the studies assumed that constraints are given prior to the experiment, and those constraints are absolute and without any ambiguity. In the normal software development practice, the availability of clustering constraints is limited due to reasons such as high cost, time constraint, out-dated software documentations, or limited background knowledge on the software to be maintained (Harman et al., 2012). For instance, domain experts who were involved in the early stage of software design might provide some constraints about the software to be maintained. However, such constraints might not be valid anymore after several phases of software updates and changes. Thus, the constraints given by the aforementioned experts might be ambiguous or contain erroneous information. In such cases, regular supervised or semi-supervised techniques discussed above cannot be used to effectively recover a high-level abstraction of software design. Hence, constrained clustering approaches that automatically generate or derive constraints from the implicit structure and behaviour of the dataset, and rely less on human effort, are more preferred in the domain of software.

To address this issue, this research proposes an approach to automatically derive ML and CL constraints from the implicit structure of the software itself based on graph theory analysis of the studied software, without feedbacks and supervision from domain experts. First, the software to be analysed is represented using a weighted complex network, followed by graph theory analysis to reveal some extra deterministic information about relationships among all the associated classes. The information is then used to support the subsequent constrained clustering approach to form cohesive clusters that are representative enough to show a high-level representation of the software design. The recovered high-level software design can act as supplementary information for software maintainers to aid in decision making when there is a request to modify or remove a particular software component. The contribution of this paper can be summarised as follows:

1. An approach to apply semi-supervised constraint clustering to aid in recovering a high-level abstraction of object-oriented software design.
2. An approach to derive clustering constraints from software systems without the feedback and supervision from domain experts.
3. An alternative solution to derive clustering constraints that helps in improving the accuracy of conventional software clustering approaches.

The paper is organized as follows: Section 2 discusses the background and related work in constrained clustering, including ways to generate and acquire constraints. Section 3 presents the proposed approach to automatically derive clustering constraints from an object-oriented software system. Section 4 presents the experimental design, along with the execution of the experiment. Section 5 gives an overall discussion based on the results obtained in the previous section. Section 6 discusses the threats to validity in this study. Finally, concluding remarks and potential future work are presented in Section 7.

2. Background and related works

Semi-supervised clustering, or commonly referred as constrained clustering has proven to be a reliable alternative to classic unsupervised approaches where a small quantity of clustering constraints is introduced in the clustering process. Clustering constraints in the form of ML and CL constraints guide the clustering algorithm into an adequate partitioning of the data, and often, improves the clustering performance significantly. Existing methods for constrained clustering fall into three categories: distance based (Bilenko and Mooney, 2003; Klein et al., 2002; Shental and Weinshall, 2003), constrained based (Davidson and Ravi, 2009; Kestler et al., 2006), and the hybrid of both.

In the domain of software, it is highly possible that software maintainers may have access to additional information about the software to be maintained, either explicitly or implicitly. For instance, domain experts or software developers who are involved in the early stages of software design or development are able to provide feedbacks to indicate whether a pair of software components should be clustered into the same functional group. This type of information, which is based on the explicit opinions and feedbacks from the domain experts, are referred as explicit clustering constraints. Domain experts often act as oracles in constrained clustering (Basu et al., 2004), where in general, a pair of clustering entities are chosen at random and presented to the oracle to judge and decide if they should or should not be grouped into the same cluster.

On the other hand, implicit information refers to some extra deterministic information about the interrelationships between software components derived from the source code itself. In various fields of research, a limited degree of side information can be revealed when performing an exploratory data analysis (Greene and Cunningham, 2007). For instance, two classes associated with inheritance relationship in object-oriented (OO) paradigm typically have stronger tendency to be grouped into the same cluster. While the given example is a straightforward one, effectively deriving implicit information from the source code requires in-depth understanding on the structure and behaviour of software systems. Software maintainers would require tool support to effectively identify and interpret the implicit information hidden in the source code because the quantity and level of granularity of the information might be too overwhelming to comprehend. The vast amount of